



AFRL-RH-FS-TR-2019-0016

Scalable Effects Simulation Environment (SESE) Version 2.5.0

Byron G. Zollars
Gabriel J. Elpers
Austin L. Goodwin
Nanohmics, Inc.

Edward A. Early
Nicholas J. Gamez
SAIC

Robert J. Thomas
**711th Human Performance Wing
Airman Systems Directorate
Bioeffects Division
Optical Radiation Bioeffects Branch**

July 2019

Final Technical Report - May 2019 to July 2019

DESTRUCTION NOTICE - Destroy by any method that will prevent disclosure of contents or reconstruction of this document.

Distribution A: Approved for public release; distribution unlimited. PA Case No: TSRL-PA-19-0206. The opinions expressed on this document, electronic or otherwise, are solely those of the author(s). They do not represent an endorsement by or the views of the United States Air Force, the Department of Defense, or the United States Government.

**Air Force Research Laboratory
711th Human Performance Wing
Airman Systems Directorate
Bioeffects Division
Optical Radiation Bioeffects Branch
JBSA Fort Sam Houston, Texas 78234**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

Qualified requestors may obtain copies of this report from the Defense Technical Information Center (DTIC) (<http://www.dtic.mil>).

"Scalable Effects Simulation Environment (SESE) Version 2.5.0"

(AFRL-RH-FS-TR- 2019-0016) has been reviewed and is approved for publication in accordance with assigned distribution statement.

Digitally signed by SHORTER.PATRICK.D.1023156390
Date: 2019.10.01 10:42:40 -05'00'

PATRICK SHORTER, Major, USAF
Chief, Optical Radiation Bioeffects Branch

MILLER.STEPHANI
E.A.1230536283

Digitally signed by
MILLER.STEPHANIE.A.1230536283
Date: 2019.10.05 20:16:50 -05'00'

STEPHANIE A. MILLER, DR-IV, DAF
Chief, Bioeffects Division
Airman Systems Directorate
711th Human Performance Wing
Air Force Research Laboratory

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute an official position of the U.S. Government.

REPORT DOCUMENTATION PAGE					<i>Form Approved</i> <i>OMB No. 0704-0188</i>	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.						
1. REPORT DATE (DD-MM-YYYY) 12-07-2019		2. REPORT TYPE Final Technical Report		3. DATES COVERED (From — To) May 2019 to July 2019		
4. TITLE AND SUBTITLE Scalable Effects Simulation Environment (SESE) Version 2.5.0				5a. CONTRACT NUMBER FA8650-19-C-6024 5b. GRANT NUMBER 5c. PROGRAM ELEMENT NUMBER 5d. PROJECT NUMBER 5e. TASK NUMBER 5f. WORK UNIT NUMBER H0S1		
6. AUTHOR(S) Byron G. Zollars, Gabriel J. Elpers, Austin L. Goodwin, Edward A. Early, Nicholas J. Gamez, and Robert J. Thomas				8. PERFORMING ORGANIZATION REPORT NUMBER 10. SPONSOR/MONITOR'S ACRONYM(S) 711 HPW/RHDO 11. SPONSOR/MONITOR'S REPORT NUMBER(S) AFRL-RH-FS-TR-2019-0016		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Research Laboratory Nanohmics, Inc. SAIC 711th Human Performance Wing 6201 E. Oltorf St 4141 Petroleum Dr Airman Systems Directorate Suite 400 JBSA Fort Sam Houston Bioeffects Division Austin, TX 78241 TX 78234 Optical Radiation Bioeffects Branch				9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory 711th Human Performance Wing Airman Systems Directorate Bioeffects Division Optical Radiation Bioeffects Branch		
12. DISTRIBUTION / AVAILABILITY STATEMENT. Distribution A: Approved for public release; distribution unlimited. PA Case No: TSRL-PA-19-0206. The opinions expressed on this document, electronic or otherwise, are solely those of the author(s). They do not represent an official position or the views of the United States Air Force, the Department of Defense, or the United States Government.						
13. SUPPLEMENTARY NOTES						
14. ABSTRACT The Scalable Effects Simulation Environment (SESE) computer program facilitates the study of the response of materials to incident laser radiation. SESE performs time-dependent simulations in a three-dimensional spatial domain and allows investigators to specify inhomogeneous material structures with arbitrary shape and constituent physical parameters. Laser radiation sources simulate the temporal dynamics of position, pointing, beam shape, and irradiance.						
15. SUBJECT TERMS						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT		18. NUMBER OF PAGES	
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U	U		102	
19a. NAME OF RESPONSIBLE PERSON Dr. Semih Kumru			19b. TELEPHONE NUMBER (include area code) 210-539-8244			

This Page Intentionally Left Blank

TABLE OF CONTENTS

Section	Page
LIST OF FIGURES	iii
LIST OF TABLES	iv
1.0 INTRODUCTION	1
1.1 Architecture	2
1.2 Input and Output Conventions	3
2.0 OBJECT AND SURFACE REPRESENTATION	4
2.1 Overview	4
2.2 Material ID files	5
2.3 Volume fraction files	5
2.4 Piecewise-linear interface calculation	6
3.0 MATERIALS AND THERMAL DAMAGE	7
3.1 Material properties	7
3.2 Damage model	9
4.0 SOURCES OF RADIATION	10
4.1 Radiation source types	10
4.1.1 Point source	10
4.1.2 Top-hat/annulus	11
4.1.3 Gaussian beam	11
4.1.4 General source distribution	12
4.2 Randomized ray initial conditions	12
4.3 Source radius of curvature	13
4.4 Initial angular spread	14
4.5 Gaussian beam propagation	14
5.0 RADIATIVE TRANSPORT MODEL	19
5.1 Overview	19
5.2 Absorption	20
5.3 Scattering	20
5.4 Reflection/refraction	21
5.5 Smart dose recalculation	23
5.6 Dose uncertainty estimation	25
5.6.1 Partition scheme	25
5.6.2 Region of interest selection	25
5.6.3 Overall uncertainty metric	26
5.6.4 Online variance calculation	26
5.6.5 Automatic termination	27
5.6.6 Parallel implementation	27
6.0 THERMAL MODEL	28
6.1 Theory and discretization	28
6.2 Domain decomposition	33
6.3 Convergence detection	35
6.4 Boundary conditions	35
7.0 ADAPTIVE TIME STEP	36

7.1	Error estimator for heat equation solver	36
7.2	Time step controller algorithm	38
7.3	Practical considerations for maximizing time step	39
8.0	VERIFICATION	40
8.1	Overview	40
8.2	Radiative transport verification	40
8.2.1	Predominantly absorbing medium ($\alpha \gg \sigma$)	40
8.2.2	Predominantly scattering medium ($\alpha \ll \sigma$)	40
8.3	Thermal diffusion verification	44
8.4	Thermal damage verification	45
8.5	Cross-validation of SESE against BTEC	46
9.0	REFERENCES	48
	APPENDIX A - INSTALLATION	50
	APPENDIX B - CONFIGURATION AND USE	58
	APPENDIX C - RUNNING SESE SIMULATIONS	68
	APPENDIX D - KEYWORD REFERENCE	90

LIST OF FIGURES

	Page
Figure 1	SESE program top-level flowchart 2
Figure 2	Cartesian computational mesh used by SESE, showing the relationship between voxel indices and voxel volume element 4
Figure 3	An example of a SESE object comprised of three materials, its representation via material ID and volume fraction 5
Figure 4	Geometry of piecewise-linear interface calculation (PLIC) plane described by surface normal unit vector $\hat{n}$ and distance α 7
Figure 5	Illustration of radiation source geometry, relative to object and computational mesh 11
Figure 6	Convergence of the overall dose uncertainty U , as computed by Equation 54, as more rays are accumulated during the Monte Carlo ray tracing step. For this graph, the uncertainty U is computed in the $M=100$ voxels with the highest dose. Rays are accumulated, the uncertainty is periodically measured after each partition, and this process continues until the desired dose uncertainty, here 0.01, is obtained 28
Figure 7	Geometric quantities in the finite volume discretization for voxels at the surface of the object 31
Figure 8	Decomposition of the computational domain into 'slabs', and MPI communications between slabs 34
Figure 9	The deposited radial energy distribution $E(R)$ computed by SESE (diamonds) versus that expected from theory (solid line), for a highly-absorbing material. The initial energy carried by each ray is set to $1/N$ for simplicity so that $E(0) = 1$. The total number of rays $N = 10^6$ 42
Figure 10	The deposited radial energy distribution $E(R)$ computed by SESE (diamonds) versus that expected from theory (solid line), for a highly-scattering material. The initial energy carried by each ray is set to $1/N$ for simplicity so that $E(0) = 1$. The total number of rays $N = 10^6$ 43
Figure 11	Temperature along x-axis, compared to theory (left) and temperature contour plot in x-y plane (right) for a separated thermal point source and point sink (thermal dipole) in a copper cube after reaching steady state conditions 45
Figure 12	Accumulated thermal damage computed as a function of time for epidermis at 55°C and 59°C (diamonds and squares, respectively), compared with Arrhenius damage integral (solid line) 46
Figure 13	Comparison of the profile of the deposited radiation dose computed with SESE (version 1.1) (lower half of plot) and BTEC (version 5.0a2) (upper half). The computations simulate a circular laser beam incident on an absorbing and scattering material 47
Figure B-1	Block diagram showing input and output files for a typical SESE simulation run 59
Figure B-2	Intersection of a cube and a sphere, generated by the above example. PLIC plane representation generated by the <code>volpoly</code> utility and visualized in Paraview 65
Figure C-1	Sketch of spherical object with planar layer, in bounding box domain 69

Figure C-2	Thermochromic damage rate as a function of temperature	73
Figure C-3	ParaView visualization of slice through full temperature field	79
Figure C-4	Comparison of probe laser axial dose at time 0 sec (dashed) and 20 sec (solid)	81
Figure C-5	Temperature at $-z$ surface of thermochromic layer, as a function of time	82
Figure C-6	Log of cumulative damage at $-z$ surface of thermochromic layer, as a function of time	82
Figure C-7	ParaView open file dialog box with converted SESE output	86
Figure C-8	Input file reader selection menu	87
Figure C-9	Data set display setup	87
Figure C-10	Object orientation and transient display	88
Figure C-11	2D slice rendering	88
Figure C-12	Visualization of multiple 2D slices of a 3D dataset	89

LIST OF TABLES

	Page
Table 1	List of material properties, their units, and their functional dependencies 8
Table A-1	Configuration script options 53
Table D-1	SESE configuration file parameters appearing in the first YAML document in the configuration file 90
Table D-2	Keys passed with <code>source</code> key, applying to all source types. 92
Table D-3	Keys passed with the <code>plot</code> key 93
Table D-4	Keys passed with the <code>repeat</code> key 93
Table D-5	Definitions of keywords used in material files to specify material properties . . 94
Table D-6	Geometric primitive classes provided for object generation. All arguments have <code>double</code> type and units of meters 95
Table D-7	Boolean region methods provided by all geometric primitives and the <code>Region</code> class 95
Table D-8	Methods provided by the <code>Object</code> class 95

1.0 INTRODUCTION

The Scalable Effects Simulation Environment (SESE) computer program facilitates the study of the response of materials to incident laser radiation. SESE performs time-dependent simulations in a three-dimensional spatial domain and allows investigators to specify inhomogeneous material structures with arbitrary shape and constituent physical parameters. Laser radiation sources simulate the temporal dynamics of position, pointing, beam shape, and irradiance.

SESE version 2.5.0 classically models the physical effects of electromagnetic radiation impinging on a complex, inhomogeneous structure. Appropriate discretization of time and space achieves the desired accuracy. The simulation performs four main functions in each time step. (1) Calculate the radiative energy distribution within the structure by tracing rays from one or more laser sources to the structure, allowing for ray-material optical interactions and energy deposition within the structure to occur. (2) Simulate the diffusion of thermal energy by solving the heat equation using an iterative algorithm, with selectable boundary conditions. (3) Track the thermodynamic state. (4) Update the structure's material properties based on the new temperature distribution, accounting for possible thermal or chemical damage.

Structures are stored in memory as a three-dimensional array of voxels with each voxel containing information regarding the physical characteristics of the material therein. Voxels comprising the boundary of the structure contain additional information that specifies a piecewise planar approximation to the structure's local shape. This method of boundary refinement provides more accurate simulation of optical phenomena such as reflection and refraction, which depend on the fine-grained details of the boundary's local surface normal vector.

Monte-Carlo radiative transport from the radiation source(s) uses ray distributions weighted probabilistically according to the spatial details of the source irradiance. Quasi-random sampling chooses ray positions to reduce variance in the computed dose distribution. In addition, the number of rays used during the radiative transport computation adjusts automatically to meet a user-specified dose uncertainty metric. Varieties of pre-defined source types are available, including Gaussian beams, whose rays propagate to simulate diffraction accurately. Rays that interact with the external and internal boundaries of the structure undergo Fresnel reflection and refraction (including total internal reflection) according to the refractive indices of the materials comprising the structure. Absorption and scattering of rays within the structure occur probabilistically, corresponding to the material's optical properties, and the user can specify a scattering phase function to model anisotropic scattering. All optical and physical material properties can have an explicit temperature dependence as well as a dependence on a damage metric that evolves with an Arrhenius rate constant, consistent with a thermally-driven chemical reaction. Energy from the radiation source(s) accumulates in each voxel for a large ensemble of source rays to obtain the radiative dose distribution.

A successive over-relaxation heat equation solver models the diffusion of the deposited energy, and accounts for inhomogeneities in both the thermal conductivity and heat capacity of the materials. A perfusion mechanism accounts for thermal sources or sinks in the bulk structure, and boundary thermal phenomena can include radiation, evaporation, convection, and conduction. Time steps are adaptive and controlled to ensure stability and bounded local errors, with a combination of explicit

and implicit stepping methods used to optimize convergence speed.

1.1 Architecture

SESE is written primarily in the C language, and it provides user flexibility by allowing a simulation to be configured with a set of input files that specify the radiation sources, material structure, and material properties. Human-readable files define the simulation parameters, desired outputs, and the time-dependent script of events. SESE runs from the command line using a configuration file as an argument. This configuration file has two parts. The first part contains all of the information necessary to specify the parameters for simulation, including references to a number of additional files that describe the object geometry, the relative placement of materials in the object, and the physical, optical, chemical, and thermodynamic properties of the materials. The second part of the configuration file describes the *action sequence* of the simulation, and includes the temporally dynamic elements of the simulation such as sources moving or turning on and off. SESE simulates the specified action sequence and outputs program status information to a journal file. It also generates a number of other output data files as specified by the user at specific times in the action sequence. Post-processing the output data files generated during a simulation provides plots or animations that represent various quantities of interest.

For complex simulations, SESE executes in a high-performance symmetric multiprocessing environment, with automatic partitioning of the various stages of computation during each time step occurring among a thread pool within a cluster of processors. A hybrid MPI/OpenMP methodology synchronizes tasks within and across processing nodes. The solution algorithm consists of solving the coupled set of radiative transfer, heat diffusion, thermodynamic evolution, and boundary modification independently over a sufficiently small time interval. A flow chart depicting the computational steps and I/O files for a SESE simulation are shown in Figure 1.

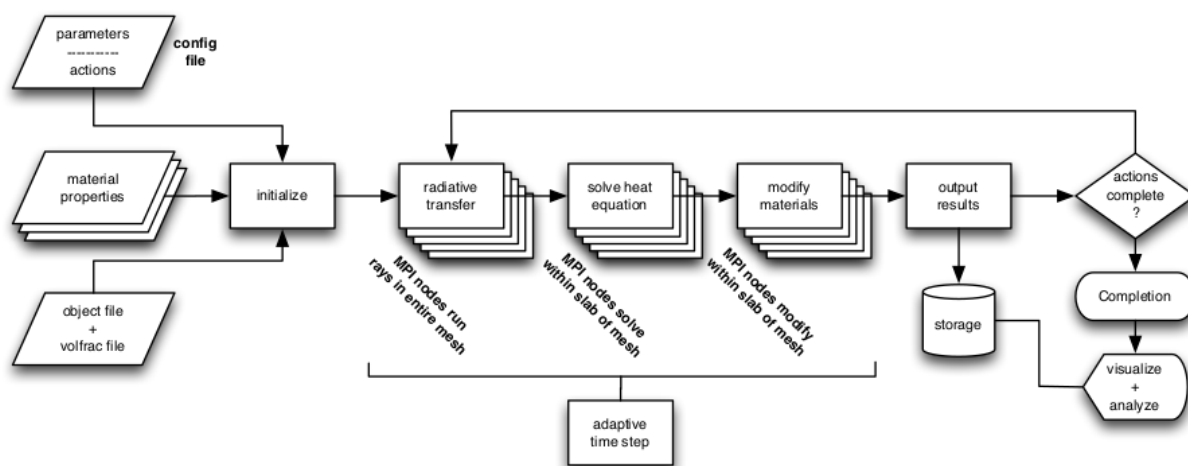


Figure 1. SESE program top-level flowchart

During initialization, input files define the simulated structure, the details of the structure's composition, and the properties of the materials that comprise it. The configuration file defines the

simulation parameters and provides an action script that describes how the simulation proceeds in time.

Radiative transport of energy from the source(s) to the structure uses a Monte Carlo ray-tracing technique to simulate absorption, scattering and reflection/refraction of light. In the radiative transport step, the computational mesh associated with the structure replicates across all available computational nodes. Each allocated thread within each of the computational nodes then independently propagates rays from the sources into the structure. Periodic analysis of the dose distribution determines achievement of the requested statistical uncertainty goal in the dose. When this occurs, the overall dose is the sum of the deposited energy across all computational nodes.

The heat equation is solved in-place in three dimensions using red-black successive over-relaxation (RBSOR) [1], [2], [3] which confers the benefit of rapid convergence and eliminates the need to assemble and store large matrices. SESE's implementation of the parallelized RBSOR technique spatially divides the structure and radiative dose distribution into slabs, one per computational node. Each node iteratively solves the heat diffusion equation within its slab, and exchanges information with its neighboring node to affect the diffusion between slabs. The iteration continues, with periodic error analysis, until achieving the desired global error metric.

Special treatment of voxels on the structure's boundaries within each slab implements heat flux source terms with the desired boundary conditions. Each computational node keeps track of boundary conditions within its slab.

Specific times specified in the action script generate and output results to data files. SESE can output various simulated field quantities, such as temperature, thermochemical damage, or radiation dose; or simulation metrics such as heat equation convergence error. Results generated are for the entire computational domain, or for any subset, including planar slices, lines, or even points. The output file flexibility simplifies the visualization and analysis of both spatial and temporal phenomena.

SESE program completion occurs when the end of the action sequence is reached – selected final results are output and execution ends. ParaView and similar visualization utilities visualize the simulation results. Further details on output file structure and typical analysis/visualization tasks are described later in this report.

1.2 Input and Output Conventions

Input files that control SESE's behavior, *e.g.* the configuration file (containing the action script), the material library file, and the individual material files, are ASCII text files formatted in YAML, a human-friendly international data serialization language (YAML Ain't Markup Language, <http://www.yaml.org/about.html>). YAML files are powerful in their ability to express data structures and associative array information in a way that both humans and computers can read and write. All SESE inputs are in SI units, with the exception of temperature, which is in degrees Celsius. Later sections cover the format and information contained in SESE input files summarized in reference tables in Appendix D.

Input and output files that convey bulk data, *e.g.* object file, volume fraction file, and data output files, are stored in a flexible binary file format called RAW3D. The RAW3D file format supports variable data set size and data type, and a number of utility programs are included with the SESE distribution. These programs convert the RAW3D files to and from text files, display the file header information (size, type, spacing, etc.), and generate XDMF (eXtensible Data Model and Format) information which can be used by HPC (High Performance Computing) data viewing and visualization tools like ParaView (<http://www.paraview.org>) and EnSight (<https://www.ensight.com>). Further information on the RAW3D format, conversion utilities, and visualizing SESE output appears in Appendix C.

2.0 OBJECT AND SURFACE REPRESENTATION

2.1 Overview

Objects and materials are discretized on a three-dimensional right-handed Cartesian grid of $n_x \times n_y \times n_z$ volume elements (*voxels*), each of size $h_x \times h_y \times h_z$, so that the full grid size is $l_x \times l_y \times l_z = n_x h_x \times n_y h_y \times n_z h_z$. During a simulation, computations take place on this same grid. It is possible to specify lower-dimension grids (*e.g.* planes, lines) by setting the number of voxels in one or two dimensions to one. Quantities within the computational domain are represented by scalar or vector quantities, called *fields*, at the discrete locations at the integer-numbered grid points i, j, k , where $0 \leq i < (n_x - 1), 0 \leq j < (n_y - 1), 0 \leq k < (n_z - 1)$. The quantity at each grid point uniformly applies within the boundaries of the voxel defined by $ih_x \leq x < (i+1)h_x, jh_y \leq y < (j+1)h_y, kh_z \leq z < (k+1)h_z$. The corner nearest the origin of the grid defines the origin of a voxel. The geometry of the grid is illustrated in Figure 2, where voxel i, j, k is shaded to help visualize the relationship between the indexed positions on the mesh and the voxels that comprise the elements of the fields.

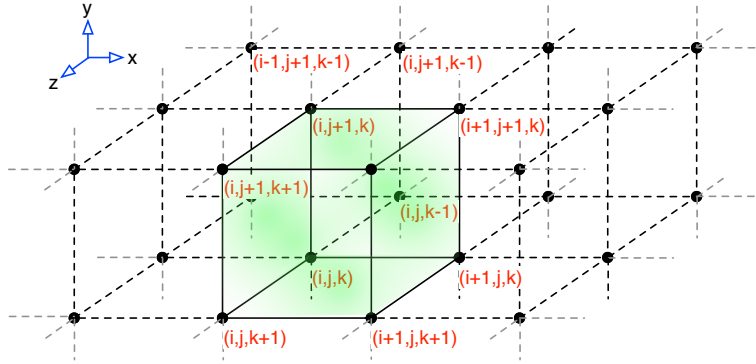


Figure 2. Cartesian computational mesh used by SESE, showing the relationship between voxel indices and voxel volume element

Two files fully specify an object: the *object* file and the *volume fraction* file. The object file contains data about *which* material occupies each voxel in the domain. The volume fraction file contains data about *how much* material fills each voxel.

2.2 Material ID Files

A file containing a three-dimensional image of the object's material composition (the object file) provides SESE the three-dimensional spatial distribution of materials within the computational mesh. This binary file, written in the SESE RAW3D data format (fully described later), contains a header followed by a serialized sequence of 16-bit unsigned integers, each corresponding to the ID of the material contained within a voxel of the domain. Each non-zero material ID corresponds to a material definition specified in a library of materials. Reserved material ID 0 denotes those voxels that are exterior to an object. Material ID 0 has a special status and does not have an associated material definition file; rather, the simulation parameters in the configuration script determine its properties. By default, the region external to the object, *i.e.* material 0, is transparent, non-thermally-conductive, and non-chemically-reactive. SESE uses the interface between a region of material ID 0 and non-zero material ID to determine the object boundary and to impose thermal boundary conditions at those interfaces.

2.3 Volume Fraction Files

An optional volume fraction file specifies how much material fills each voxel. Inside the object, the volume fraction is unity; outside, it is zero; in voxels containing the object surface, it is between 0 and 1.

This volume fraction field reconstructs the surface locally using a plane lying within each surface voxel, with location and orientation inferred from the volume fraction field. This extra information enables a much more faithful representation of curved or sloped object surfaces on the voxel grid. Simulated radiation interacts with the local surface planes to produce refractive and reflective effects, and the thermal solver utilizes the local exposed planar surface area to simulate thermal losses at the surface. Figure 3 illustrates the relationship between the object and the information contained in these two files.

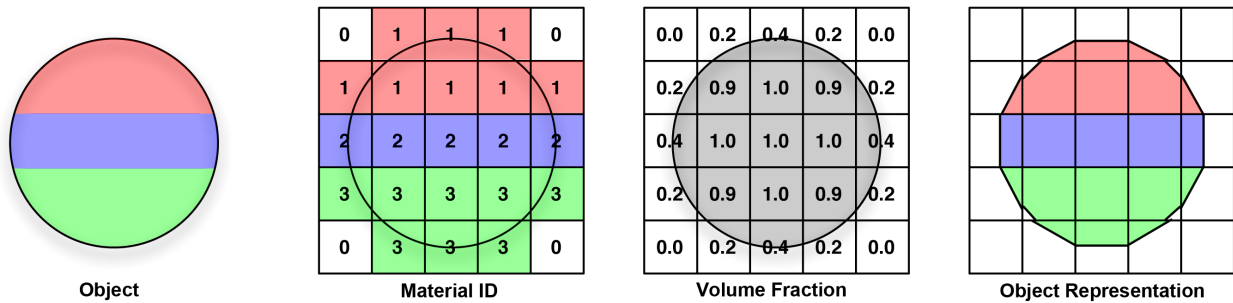


Figure 3. An example of a SESE object comprised of three materials, its representation via material ID and volume fraction

If the volume fraction file is not specified, SESE automatically generates a volume fraction field with all surface voxels (voxels with a neighbor having material ID 0) set to $\frac{1}{2}$. However, this typically results in a poor surface reconstruction with many 'gaps' and discontinuities for any surface except those with plane orientations (100) or (110) with respect to the voxel grid. Since not specifying a volume fraction file may lead to macroscopically incorrect surface thermal losses and optical refraction, proper generation and utilization of the volume fraction file is very beneficial.

Utilities supplied with the SESE distribution assist with the creation and visualization of material ID and volume fraction files.

2.4 Piecewise-Linear Interface Calculation

The volume fraction field f can be used to reconstruct a piecewise planar approximation of the interface. Within each surface voxel (defined as being adjacent to a voxel containing material ID 0), the interface is represented by a plane containing the locus of points r that satisfy the equation

$$\hat{n} \cdot r = \alpha \quad (1)$$

where $\hat{n}$ is the outward unit normal vector and α is the distance of the plane from the origin of the voxel, i.e. from the corner represented by the grid coordinates i, j, k . The normal vector is parallel to the *material gradient* ∇f , which is estimated numerically in a $3 \times 3 \times 3$ domain surrounding voxel i, j, k by the method of [4]:

$$\left. \frac{\partial f}{\partial x} \right|_{i,j,k} \approx \frac{F_{i,+} - F_{i,-}}{2} \quad (2a)$$

$$F_{i,+} = \frac{1}{6}(2f_{i+1,j,k} + f_{i+1,j-1,k} + f_{i+1,j+1,k} + f_{i+1,j,k-1} + f_{i+1,j,k+1}) \quad (2b)$$

$$F_{i,-} = \frac{1}{6}(2f_{i-1,j,k} + f_{i-1,j-1,k} + f_{i-1,j+1,k} + f_{i-1,j,k-1} + f_{i-1,j,k+1}) \quad (2c)$$

where $f_{i,j,k}$ is the volume fraction of the voxel at the coordinate i, j, k . Similar formulas hold for the y - and z -components of the material gradient. The normal vector of the local planar interface is

$$\hat{n} = -\nabla f / \|\nabla f\| \quad (3)$$

where $\|\cdot\|$ denotes the Euclidian norm. After the normal vector is determined, the plane translates along its normal until the interface cuts a fraction f of the voxel volume. This translation uniquely determines the offset α . This method of local interface reconstruction is the piecewise-linear interface calculation (PLIC). This geometry is illustrated in Figure 4.

For a surface voxel with normal vector $\hat{n} = (n_x, n_y, n_z)$, dimensions $h = (h_x, h_y, h_z)$ and interface distance from the voxel origin α , the volume fraction f can be calculated using the inclusion-exclusion principle with the volumes of similar tetrahedra [5]:

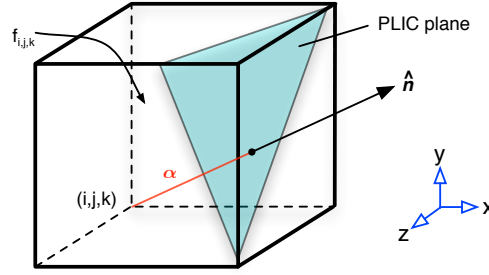


Figure 4. Geometry of piecewise-linear interface calculation (PLIC) plane described by surface normal unit vector $\hat{n}$ and distance α

$$f = \frac{1}{6n_x n_y n_z h_x h_y h_z} \left(\alpha^3 - \sum_i B(\alpha - n_i h_i) + \sum_i B(\alpha - \alpha_{\max} + n_i h_i) \right) \quad (4a)$$

$$\alpha_{\max} = \sum_i n_i h_i \quad (4b)$$

$$B(x) = \begin{cases} x^3, & x > 0 \\ 0, & x \leq 0 \end{cases} \quad (4c)$$

For some plane orientations, this formula relies on the precise cancellation of large quantities, and thus by itself is subject to roundoff errors. The roundoff error can be reduced by deriving analytic reductions of the general formula to deal with each region of behavior properly, as detailed by [5].

The inverse problem of computing α given $\hat{n}$, h and f also has a numerically stable piecewise implementation. Scardovelli and Zaleski's approach to the plane calculation, implemented in SESE, takes advantage of symmetries (such as performing transformations so that the normal vector always lies in the first octant) and exploiting the odd symmetry of the function that relates f and α to simplify the programmatic implementation. The full details of this inverse calculation are given in [6].

3.0 MATERIALS AND THERMAL DAMAGE

3.1 Material Properties

Materials in SESE simulations may have dynamic physical, optical, thermodynamic, and chemical properties that depend on the wavelength, the local temperature, and thermally-induced chemical *damage* caused by chemical or physiochemical reactions. The corresponding material definition file specifies these properties for each material, which consists of several sections for optical, thermal, and chemical properties. Materials properties are specified in SI units, except for temperature, which is specified in degrees Celsius.

The list of material properties used in SESE's calculations are given in Table 1. Most material

properties can be temperature and damage-dependent; optical properties may additionally depend on the wavelength of incident light.

Each of the ‘native’ and fully ‘damaged’ states specify material properties independently as functions of temperature T . For partially-damaged materials, the material properties are interpolated linearly between these states depending on the relative concentrations of native (c) and damaged material ($1 - c$). For example, SESE interpolates the thermal conductivity for partially damaged material between the native and damaged thermal conductivities κ_{native} and κ_{damaged} , respectively.

$$\kappa(T, c) = c \cdot \kappa_{\text{native}}(T) + (1 - c) \cdot \kappa_{\text{damaged}}(T) \quad (5)$$

Table 1. List of material properties, their units, and their functional dependencies

Property	Units	Depends on
optical		
index of refraction		T, f, λ
linear optical absorption coefficient	m^{-1}	T, f, λ
linear optical scattering coefficient	m^{-1}	T, f, λ
optical scattering anisotropy $g \in [-1, 1]$		T, f, λ
thermal		
density	kg/m^3	T, f
specific heat capacity	$\text{J}/\text{kg} \cdot \text{K}$	T, f
thermal conductivity	$\text{W}/\text{m} \cdot \text{K}$	T, f
perfusion rate	s^{-1}	T, f
metabolic heat generation	W/m^3	T, f
surface emissivity		T, f
convective heat transfer coefficient	$\text{W}/\text{m}^2 \cdot \text{K}$	T, f
evaporative heat transfer coefficient	$\text{W}/\text{m}^2 \cdot \text{Pa}$	T, f
chemical		
Arrhenius prefactor	s^{-1}	T
reaction activation energy	J/mol	T

3.2 Damage Model

Dynamic changes in physical properties of laser-irradiated materials can occur because of many complex chemical and physical processes, which may be reversible or irreversible (for further details, see [7]). The relevant processes may include:

1. temperature dependence of optical properties;
2. chemical reaction with conversion to other species;
3. conformational changes;
4. dehydration.

The Arrhenius equation models reactions leading to irreversible damage in SESE. The model assumes that chemical reactions occur at a rate k that depends on the time-dependent temperature $T(t)$ according to the Arrhenius reaction model.

$$k = A \exp\left(-\frac{E_a}{RT(t)}\right) \quad (6)$$

where E_a is the activation energy, A is the Arrhenius prefactor, and R is the gas constant (8.314 J/mol/°C). Typical ranges of values for a material like biological tissue are $A = 10^{50}$ – 10^{105} sec⁻¹ and $E_a = 10^5$ – 10^6 J/mol; these parameters vary widely for different materials. Although damage may be the result of several very different processes, a simplified model is used in SESE, in which possibly multiple reactions occurring within a single material are approximated by a single representative ‘damage’ reaction with representative parameters A and E_a , both of which may depend upon temperature.

The physical properties of the material depend on both the temperature and the damage history. SESE linearly interpolates these properties between the ‘native’ and ‘damaged’ values, depending on the relative concentration of each type of material. For example, the absorption coefficient of a partially damaged material at time t is

$$\alpha(T, t) = C(t) \cdot \alpha_{\text{native}}(T(t)) + (1 - C(t)) \cdot \alpha_{\text{damaged}}(T(t)) \quad (7)$$

where $C(t)$ is the fraction of native material remaining, and is related to the accumulated damage Ω

$$\Omega(t) = -\ln(C(t)) = \int_0^t A \exp\left(-\frac{E_a}{RT(t')}\right) dt' \quad (8)$$

In every material voxel, the accumulated damage Ω increases with time and SESE interpolates the voxel's material properties between their damaged and undamaged values accordingly.

$$\alpha(T, t) = e^{-\Omega(t)} \cdot \alpha_{\text{native}}(T(t)) + (1 - e^{-\Omega(t)}) \cdot \alpha_{\text{damaged}}(T(t)) \quad (9)$$

After each thermal solution step, the trapezoidal rule locally updates the accumulated damage Ω in each voxel:

$$\Omega(t + \Delta t) = \Omega(t) + \int_t^{t+\Delta t} A \exp\left(-\frac{E_a}{RT(t')}\right) dt' \quad (10a)$$

$$\approx \Omega(t) + \frac{\Delta t}{2} \left[A(T(t)) \exp\left(-\frac{E_a(T(t))}{RT(t)}\right) + A(T(t + \Delta t)) \exp\left(-\frac{E_a(T(t + \Delta t))}{RT(t + \Delta t)}\right) \right] \quad (10b)$$

4.0 SOURCES OF RADIATION

SESE can simulate the effects of radiation incident on an object originating from one or more sources. The position and orientation of each source is unconstrained, can be located outside the computational mesh, and can dynamically translate and change angle during the simulation via commands in the action script. In general, the user specifies the position, angular orientation and irradiance distribution of the source, including the option of selecting from one of a number of pre-defined, often-used source types. Alternately, the user can specify the two-dimensional irradiance distribution of the source with arbitrary spatial detail.

SESE samples ray initial positions from a planar irradiance distribution, referred to as the source plane. The source plane has Cartesian coordinates u and v , and the orientation of the source plane is specified relative to the x, y, z plane holding the computational mesh and the object being irradiated. Varieties of primitive source types are available for use by specifying their parameters directly in the configuration file. Alternately, an external file specifies sources with an arbitrary two-dimensional spatial distribution. The primitive source types can use quasi-random sampling, which has superior radiative dose convergence properties and can greatly improve the efficiency of the Monte Carlo simulation. Several common source distributions are available in SESE, including point, top-hat/annulus and Gaussian beam sources. A depiction of two radiation source planes and their placement relative to the computational mesh is shown in Figure 5. Details of the radiation source specification are located in Section D.3.

4.1 Radiation Source Types

4.1.1 Point source.

The isotropic point source causes all rays to originate from a single point with a uniformly randomized initial direction.

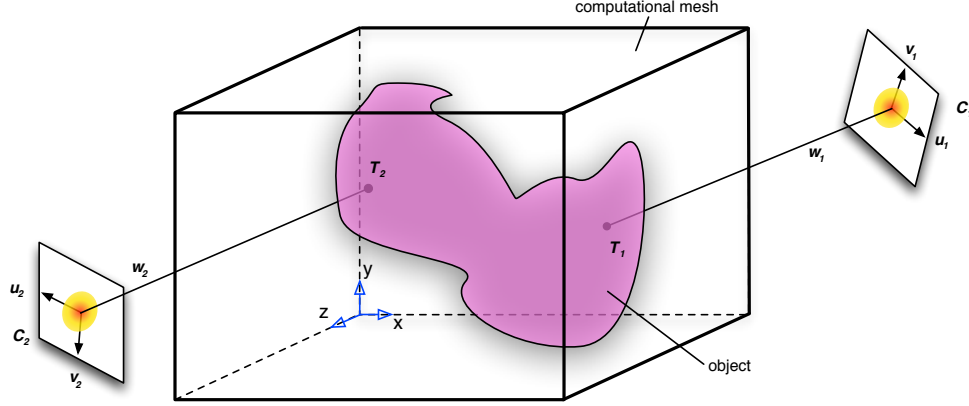


Figure 5. Illustration of radiation source geometry, relative to object and computational mesh

4.1.2 Top-hat/annulus.

A top-hat beam source has a uniform irradiance in an elliptical region of the source plane, and zero irradiance elsewhere. The radial extents of the top-hat beam in two orthogonal directions u and v are independent, allowing an elliptical beam profile. The radii of curvature of the beam in the source plane are also in the same two orthogonal directions; astigmatism is possible by specifying different curvatures for each axis. SESE uses the convention that positive radii of curvature have a center of curvature located in the direction of the beam's propagation. Thus, a positive radius of curvature would cause a convergence of the beam's rays as the beam propagates. The top-hat source can optionally be partially obscured by specifying the location of an elliptical obscuration (relative to the beam center), creating an annular irradiance distribution.

For this source type, a simple rejection method with uniform pseudo-random or quasi-random samples generates starting ray positions. Rays are propagated using geometric optical ray tracing from their point of origin within the source plane. Certain combinations of parameters may result in geometric line or point singularities. Geometric ray tracing does not support the simulation of diffraction phenomena.

4.1.3 Gaussian beam.

For Gaussian beams, the joint distribution of initial ray positions for each Cartesian component in the source plane factors into the marginal distributions for each component. Hence, the initial position of a ray in each coordinate of the uv source plane is sampled independently using the inverse transform method. That is, SESE samples two random locations ξ_1 and ξ_2 on the unit square $(0, 1)^2$ and then maps each to the cumulative distribution function (CDF) Φ of the normal distribution as in

$$u = \Phi^{-1}(\xi_1; u_0, \sigma_u^2); v = \Phi^{-1}(\xi_2; v_0, \sigma_v^2) \quad (11)$$

where (u_0, v_0) is the location of the center of the beam and σ_u, σ_v are the standard deviations of the beam irradiance in the u and v directions, respectively, as specified by the user.

SESE uses a special method for Gaussian beam propagation that accounts for diffractive effects near the beam waist. Further details are given in Section 4.5.

4.1.4 General source distribution.

In addition to the above basic source types, SESE can sample from an arbitrary two-dimensional irradiance distribution specified in an external file. SESE samples from this arbitrary distribution by serializing the two-dimensional data and using the method of [8] to generate samples efficiently from the given distribution. This method is compatible only with the pseudo-random number generator, not with quasi-random sampling.

4.2 Randomized Ray Initial Conditions

A Mersenne Twister 19937 generator [9] provides pseudo-random samples $\xi \in (0, 1)$. The pseudo-random generator generates nearly all random numbers used in the Monte Carlo calculation, including scattering lengths, scattering angles, random beam spread angles, and ray starting positions. Each parallel MPI node initializes with a unique seed, calculated as the current UNIX timestamp plus the node's MPI rank, ensuring independence of samples in parallelized simulations.

Alternatively, for certain source distributions, ray starting positions may be sampled from a quasi-random number generator. Quasi-random sampling, in contrast to pseudo-random sampling, is a method that generates apparently random samples using a low-discrepancy sequence [10]. The samples generated appear to be randomly distributed, but in fact they cover subsets of the sampled domain more uniformly than truly random samples. This results in faster convergence of the Monte Carlo method. Since the distribution of absorbed radiation is typically highly spatially correlated with an extended incident source distribution, such as a Gaussian or top-hat beam, SESE applies quasi-random sampling to the two-dimensional distribution of initial ray positions for these source types.

It is important to note that the quasi-random number generator is deterministic in nature and cannot be seeded. To prevent the simulation from sampling the same positions in every simulation, pseudo-random shifts v_i , sampled from the pseudo-random number generator at the beginning of the simulation, are added to each component ξ_i of the quasi-randomly sampled point. The result maps back to the unit interval:

$$\xi'_i = (\xi_i + v_i) \bmod 1.0 \quad (12)$$

A parallel application making use of a quasi-random generator requires additional precautions. Suppose that N parallel nodes need to generate distinct quasi-random samples without communication. If each node naively samples the same quasi-random sequence of points from the unit square $(0, 1)^2$, all nodes will launch rays from starting positions that are perfectly correlated (dif-

fering only by the pseudo-random offset), with the undesirable consequence that results computed by different nodes may be perfectly correlated rather than independent.

The approach of [11] avoids the difficulty, which treats the parallelization as adding an extra dimension to the sampling domain. Consider a sample generated in the computational node with rank n , where n ranges from 0 to $N - 1$, and N being the number of computational nodes. To sample a pair of random numbers in node n , three quasi-random samples (ξ_1, ξ_2, ξ_3) are repeatedly generated using a Halton sequence [12] in the unit cube $(0, 1)^3$ until the coordinate ξ_3 satisfies

$$n/N < \xi_3 < (n + 1)/N \quad (13)$$

When this criterion is satisfied, the sample $(\xi_1, \xi_2) \in (0, 1)^2$ is accepted. This effectively partitions the unit cube into N slabs, assigns one slab to each node, and the sequence of samples that fall within each slab is ‘projected’ onto the unit square $(0, 1)^2$. This projected sequence is also quasi-randomly distributed in the unit square. Each node thereby produces a distinct sequence of samples quasi-randomly distributed in the unit square; furthermore, the union of samples from all nodes is also quasi-randomly distributed.

4.3 Source Radius Of Curvature

Coordinates in the source plane are defined in the uv Cartesian plane, with the beam’s radiation propagating primarily in the direction

$$\hat{s} = \hat{u} \times \hat{v} \quad (14)$$

For a beam with radii of curvature R_u and R_v , defined in the source plane ($s = 0$) in the (u, v, s) coordinate system), ray initial directions will be a function of their source plane coordinates. For a ray with starting position $(u_0, v_0, 0)$, the convention that a positive radius of curvature corresponds to a center of curvature in the direction of propagation is used. In the paraxial approximation, the phase of the wavefront is

$$\phi(u, v, s) = k \left(s - \frac{u^2}{2R_u} - \frac{v^2}{2R_v} \right) + \phi_0, \quad (15)$$

where $k = 2\pi/\lambda$ is the vacuum wavenumber and ϕ_0 is an arbitrary phase offset. The wavefront passing through the source plane $s = 0$ is thus a paraboloid with the local normal vector

$$\hat{n} = \frac{\nabla \phi}{\|\nabla \phi\|} = \frac{\langle -u/R_u, -v/R_v, 1 \rangle}{\sqrt{(u/R_u)^2 + (v/R_v)^2 + 1}} \quad (16)$$

If a ray is launched from some starting position with its direction parallel to this normal vector, its trajectory $\langle u, v, s \rangle$ will pass through the plane $u = 0$ when $s = R_u$, and will likewise pass through the plane $v = 0$ when $s = R_v$. Thus, the ray trajectories will focus into zero-width lines along the v -axis and the u -axis, respectively, at distances equal to the radii of curvature in each axis. If $R_u = R_v$, the ray trajectories will converge to a point focus at the location $s = R_{u,v}$; thus, diffraction near the focal point is not accounted for in the geometrical ray-tracing for non-Gaussian beam sources.

4.4 Initial Angular Spread

Any of the source types, except the Gaussian beam, can have an optional random angular spread. This angular spread modifies the initial direction $\hat{n}$ of the ray, determined by the beam radii of curvature, by the addition of a (usually) small random angular offset. A ‘cone’ spread results in emission into a specified range of polar angle θ uniformly about $\hat{n}$. A ‘cosine’ spread results in the addition of an emission angle according to the angle probability distribution $\cos^p \theta$, where p is an optional exponent. Larger values of p tend to make the emission more heavily weighted in the initial ray direction $\hat{n}$. The angular spread is a somewhat crude approximation to diffraction behavior. For example, a cone angular spread of λ/D is added to a radiation source with radius of curvature $R_u = R_v = L$, where λ is the source wavelength and D is the diameter of the source in the source plane. Then, instead of forming a point focus at $s = L$, the rays have a circular distribution with diameter $\lambda L/D$.

4.5 Gaussian Beam Propagation

A special method traces rays in a Gaussian beam, which more accurately captures the irradiance near the beam waist due to diffraction. Typically, Monte Carlo simulations of radiative transfer from a laser source assume that rays propagate in a straight line from source to target. As described in Section 4.3, defining two independent radii of curvature in the source plane focuses a simulated beam astigmatically. When a ray launches from its starting position in the source plane, the initial ray direction is normal to the local wavefront (equiphase surface). If rays from a source with a positive radius of curvature propagate geometrically (i.e. in straight lines along their initial directions), their trajectories will intersect, forming either a line focus or a point focus.

Actual beams exhibit diffractive effects that cause the beam to come to a focus with finite extent. Far from focus, the irradiance distribution in a plane transverse to the propagation direction is nearly identical to that produced by geometric propagation; however, in certain cases, it is useful to simulate the physical effects of beam propagation due to diffraction accurately. This is particularly true in geometries where simulations take place close to a focal region of the beam, or when the value of the Fresnel number is close to one. The Fresnel number is

$$N_F = \frac{w^2}{\lambda L} \quad (17)$$

where w is the radius of the transmitted beam, λ is its wavelength, and L is the propagation distance. The Fresnel number is dimensionless, and characterizes the importance of diffraction in any beam propagation scenario. Beam propagation with values of $N_F \leq 1$ have pronounced physical

effects due to diffraction.

While the general case of Gaussian beam propagation is a three-dimensional problem, we first introduce the Gaussian beam formalism along a single transverse axis for simplicity. The well-known TEM₀₀ Gaussian beam solution is the lowest-order solution to the Helmholtz equation in the paraxial approximation. Along a single transverse axis (u , say), the complex electric field amplitude of a monochromatic, uniformly polarized beam propagating in the s direction takes the form

$$E(u, s) \propto \exp \left(-\frac{u^2}{w(s)^2} + iks + ik\frac{u^2}{2R(s)} + i\zeta(s) \right) \quad (18)$$

where k is the wavenumber, i is the unit imaginary number, $w(s)$ is the beam half-width (the distance at which the field amplitude drops to $1/e$ of its on-axis value, or its irradiance drops to $1/e^2$ of its on-axis value), $R(s)$ is the radius of curvature of the beam wavefront, and $\zeta(s)$ is an additional phase shift known as the Gouy phase. Suppose that the minimum beam half-width $w_0 = w(s_0)$ occurs at the location s_0 . The quantities w_0 , s_0 , along with the wavelength λ , provide sufficient information to determine the quantities $w(s)$ and $R(s)$ everywhere via the relations presented below. Defining the quantity $s_R = \pi w_0^2 / \lambda$, known as the Rayleigh range, the beam half-width and radius of curvature at any other s location are

$$w(s)^2 = w_0^2 \left(1 + \left(\frac{s - s_0}{s_R} \right)^2 \right) \quad (19a)$$

$$R(s) = -(s - s_0) \left(1 + \left(\frac{s_R}{s - s_0} \right)^2 \right) \quad (19b)$$

Observe that the relation for the beam half-width $w(s)$ defines a hyperbola in the us plane that corresponds to the locus of points at which the field amplitude drops to $1/e$ of its on-axis value. The corresponding irradiance distribution along the u -axis is

$$I(u, s) = |E(u, s)|^2 \propto \exp \left(-2\frac{u^2}{w(s)^2} \right) \quad (20)$$

In SESE, it is more convenient to specify the beam half-width $w(0)$ and radii of curvature $R(0)$ in the source plane than to specify the minimum half-width (also called the *beam waist*) w_0 , the beam waist location s_0 , and the Rayleigh range s_R directly. From these quantities in the source plane, the beam waist location s_0 and the Rayleigh range s_R are as follows:

$$s_0 = \frac{R(0)}{1 + \left(\frac{\lambda R(0)}{\pi w(0)^2}\right)^2} \quad (21a)$$

$$s_R = \left(\frac{\lambda R(0)}{\pi w(0)^2}\right) s_0 \quad (21b)$$

Note the implication of the above equation for s_0 – the location of the beam waist s_0 is *not* coincident with the focal point (at $s = R(0)$) if the quantity

$$\left(\frac{\lambda R(0)}{\pi w(0)^2}\right)^2 = \left(\frac{1}{\pi N_F}\right)^2 \geq 0.1 \quad (22)$$

In three-dimensions, for the case of an astigmatic Gaussian beam, the paraxial wave equation is separable, admitting solutions that are products of independent one-dimensional solutions in u and v . The propagation is then decoupled along the u and v axes and the electric field amplitude is as follows:

$$E(u, v, s) \propto \exp\left(-\left(\frac{u^2}{w_u(s)^2} + \frac{v^2}{w_v(s)^2}\right) + iks + ik\left(\frac{u^2}{2R_u(s)} + \frac{v^2}{2R_v(s)}\right) + i\zeta(s)\right) \quad (23)$$

where there are now independent beam half-widths $w_u(s)$, $w_v(s)$ and radii of curvature $R_u(s)$, $R_v(s)$ for the transverse axes, satisfying

$$w_u(s)^2 = w_{0,u}^2 \left(1 + \left(\frac{s - s_{0,u}}{s_{R,u}}\right)^2\right) \quad (24a)$$

$$R_u(s) = -(s - s_{0,u}) \left(1 + \left(\frac{s_{R,u}}{s - s_{0,u}}\right)^2\right) \quad (24b)$$

$$w_v(s)^2 = w_{0,v}^2 \left(1 + \left(\frac{s - s_{0,v}}{s_{R,v}}\right)^2\right) \quad (24c)$$

$$R_v(s) = -(s - s_{0,v}) \left(1 + \left(\frac{s_{R,v}}{s - s_{0,v}}\right)^2\right) \quad (24d)$$

In these relations, $w_{0,u} = w_u(s_{0,u})$ is the beam waist in the u direction, $w_{0,v} = w_v(s_{0,v})$ is the beam waist in the v direction, and $s_{0,u}$, $s_{0,v}$ are the locations of these waists, respectively. The irradiance now has the two-dimensional Gaussian profile:

$$I(u, v, s) = |E(u, v, s)|^2 \propto \exp \left(-2 \left(\frac{u^2}{w_u(s)^2} + \frac{v^2}{w_v(s)^2} \right) \right) \quad (25)$$

The following observations regarding the Gaussian beam solution:

1. The beam irradiance in a plane transverse to the propagation direction maintains an elliptical Gaussian transverse profile with $1/e^2$ irradiance radii $w_u(s), w_v(s)$ at every point s along the beam axis, although the size and eccentricity of this profile may vary with s .
2. The shape of the Gaussian irradiance profile in each plane is defined completely by the beam half-widths $w_u(s)$ and $w_v(s)$, which describe hyperbolae in the us and vs planes, respectively.

In SESE's Monte Carlo radiative transfer algorithm, rays represent bundles of radiant energy along a Poynting vector. Our approach to ray tracing with a Gaussian beam, therefore, is the following:

- Rays are launched from their starting positions in the source plane according to a two-dimensional Gaussian distribution with $1/e^2$ irradiance radii $w_u(0), w_v(0)$.
- Thereafter, each ray is stepped incrementally by an appropriate distance δ after being assigned an updated direction vector

$$v(u, v, s) \propto \langle -u/R_u(s), -v/R_v(s), 1 \rangle \quad (26)$$

which is perpendicular to the local wavefront computed by evolving the R_u and R_v at the position s .

If a ray is launched within the us plane, its trajectory will follow one of the hyperbolae in the family proportional to the hyperbola $w_u(s)$; i.e.

$$\frac{u}{u_0} = \frac{w_u(s)}{w_u(0)} \quad (27a)$$

$$\left(\frac{w_u(s)}{w_{0,u}} \right)^2 - \left(\frac{s - s_{0,u}}{s_{R,u}} \right)^2 = 1 \quad (27b)$$

and similarly for a ray launched within the vs plane. Conserving the number of rays also conserves the beam 'energy', so the beam irradiance (proportional to the transverse ray density) naturally increases as rays converge near the waist. To demonstrate that this technique gives rise to the correct beam profile everywhere in the beam, it is sufficient to demonstrate the following:

1. The transverse beam irradiance profile indeed has an elliptical Gaussian distribution aligned with the u, v directions in every plane s ; and
2. This Gaussian profile has the correct beam widths $w_u(s), w_v(s)$ at each s .

The first statement follows from the fact that as a ray propagates by a small step δ from s to $s' = s + \Delta s$ along the ray direction unit vector $\hat{v}(u, v, s)$, its coordinates change as follows:

$$s' = s + v_s \delta = s + \Delta s \Rightarrow \delta = \frac{\Delta s}{v_s} \quad (28a)$$

$$u' = u + v_u \delta = u + \left(-\frac{u}{R_u} \Delta s \right) = u \left(1 - \frac{\Delta s}{R_u} \right) \quad (28b)$$

$$v' = v + v_v \delta = v + \left(-\frac{v}{R_v} \Delta s \right) = v \left(1 - \frac{\Delta s}{R_v} \right) \quad (28c)$$

The distribution of rays over (u', v') in the plane $s' = s + \Delta s$ is a simple linear scaling of the coordinates of the distribution over (u, v) in the plane s . Therefore, if the initial ray distribution is Gaussian, the distribution after the linear scaling of the coordinates will be Gaussian also, with the same alignment along the u and v axes. This result holds regardless of the choice of δ . The second statement follows from the fact that the ray direction is normal to the paraboloidal wavefront at each step. For a ray originating at $\{w_u(0), 0, 0\}$, propagation by small steps δ along the vector

$$v(u, 0, s) \propto \langle -u/R_u(s), 0, 1 \rangle \quad (29)$$

is equivalent to numerical integration of the curve

$$u'(s) = -\frac{1}{R_u(s)} u(s) \quad (30)$$

It may be easily verified analytically that $w_u(s)$ is the solution to this integration. The same argument holds for the beam waist in the v coordinate. The accuracy of the integration depends on the step size δ ; if δ is too large, then rays will overshoot their trajectories near the waist and will tend to concentrate along the beam axis. Currently, the implementation of this integration scheme in SESE is equivalent to forward Euler integration, with a step size chosen as a small fraction of the minimum radius of curvature of the ray trajectory

$$\delta < \varepsilon \min \{ |1/u''(s)|, |1/v''(s)| \} \quad (31)$$

where the radii of curvature of the ray trajectory coordinates are

$$|1/u''(s)| = \frac{s_{R,u}^2}{|u(s)|} \left(1 + \left(\frac{s - s_{0,u}}{s_{R,u}} \right)^2 \right)^2 \quad (32a)$$

$$|1/v''(s)| = \frac{s_{R,v}^2}{|v(s)|} \left(1 + \left(\frac{s - s_{0,v}}{s_{R,v}} \right)^2 \right)^2 \quad (32b)$$

This choice allows δ to become large in the regions far from the beam waist, where straight lines will approximate the ray paths, but finely resolves the path near the beam waist where the velocity vector changes significantly. Future versions may implement a more refined integration scheme.

SESE's Monte Carlo radiative transfer calculation uses this Gaussian beam propagation algorithm. Each ray starts at a random position in the source plane sampled from the Gaussian source irradiance distribution. The ray then propagates through the media with a variable step size to ensure accurate following of the beam through the region(s) containing the beam waist(s). If a scattering or reflection/refraction event occurs, the ray continues with ordinary geometric propagation from that point.

5.0 RADIATIVE TRANSPORT MODEL

5.1 Overview

The Monte Carlo method for the transport of radiative energy is a statistical method for determining the spatial distribution of radiation absorbed within an object, the *radiative dose*, having SI units of W/m³. Incident radiation is modeled by an ensemble of small, discrete, ballistic packets of energy, called 'rays', which are emitted by a source and thereafter interact with the object via absorption, scattering, reflection and refraction. Rays have an energy 'weight' and a vector propagation direction.

SESE's radiative transport algorithm sets each initial ray position by sampling from the probability distribution corresponding to a planar source irradiance distribution. It then propagates the ray forward along a direction vector determined by the source radiance. The ray interacts with the material in each voxel of the object it encounters, according to the material's optical properties, changing direction (scattering, reflecting, and refracting) and depositing energy (absorption) as it propagates. Propagation continues until one of the following occurs:

- the ray reaches a voxel wall;
- the ray reaches a PLIC plane;
- the ray experiences a scattering event.

At a voxel wall or PLIC plane, the ray splits into a reflected ray and a refracted ray if there is a contrast in the refractive index across the interface. At a scattering event, a scattering angle sampled from the local material's scattering phase function determines the new ray direction vector.

5.2 Absorption

Each ray carries a weight w , initially unity, which is attenuated exponentially with path length l as the ray traverses a region containing material with absorption coefficient α :

$$w' = w \exp(-\alpha l) \quad (33)$$

During propagation, the weight lost by a ray $\Delta w = w - w'$ is gained by the voxel through which the ray has passed. Intercepting rays as they cross voxel walls ensures deposition of the correct amount of energy in each voxel. Ray propagation continues until the ray weight falls below a user-defined threshold, at which point the ray terminates, with the remaining weight delivered to the voxel containing the ray.

After this process has propagated many rays, the weight deposited in each voxel is divided by the total number of rays N propagated, multiplied by the total source power P and divided by the local volume of material contained in the voxel, to obtain the power density of radiation absorbed by the voxel:

$$Q = w \cdot \frac{P}{N f h_x h_y h_z} \quad (34)$$

where f is the local volume fraction. The quantity Q has dimensions of energy per unit volume per unit time. Q is the 'dose', and serves as a source term in the heat equation.

5.3 Scattering

During propagation, SESE samples the distance ζ to the next scattering event from an exponential distribution:

$$p(\zeta) = \sigma \exp(-\sigma \zeta) \quad (35)$$

where $p(\zeta)d\zeta$ is the probability that a scattering event will occur in the interval $[\zeta, \zeta + d\zeta)$ and σ is the scattering coefficient of the material. When a ray propagates to a scattering event, SESE samples a scattering angle from the local material's scattering phase function and a new ray direction vector is calculated. Currently, SESE samples the scattering angle from the Henyey-Greenstein phase function.

The Henyey-Greenstein scattering phase function is a phenomenological distribution used to describe anisotropic scattering in turbid optical media [13]. It originally described the anisotropic scattering of light through interstellar dust but has recently found applicability in biomedical studies of radiation propagation through tissue [14]. The functional form is

$$p(\theta) = \frac{1}{4\pi} \frac{1 - g^2}{(1 + g^2 - 2g \cos \theta)^{3/2}} \quad (36)$$

and with this normalization over 4π solid angle it may be considered as the probability density for a ray being deflected by an angle θ during a given scattering event. The parameter $g \in [-1, 1]$ is called the ‘anisotropy parameter’, where $g = 0$ for isotropic scattering and $g = 1$ or -1 for strongly forward or backward scattering, respectively.

It is often more useful in practice to rewrite the function $p(\theta)$ as a probability distribution for the random variable $\mu = \cos \theta$:

$$p(\mu) = \frac{1}{2} \frac{1 - g^2}{(1 + g^2 - 2g\mu)^{3/2}} \quad (37)$$

which has the cumulative distribution function (CDF)

$$CDF(\mu) = \frac{1 - g^2}{2g} \left[(1 + g^2 - 2g\mu)^{-1/2} - (1 + g^2)^{-1/2} \right] \quad (38)$$

In SESE, scattering angle cosines μ are sampled by generating a random number $\xi \in [0, 1)$ and letting $CDF(\mu) = \xi$. Analytically inverting the cumulative distribution yields the cosine of the scattering angle

$$\mu = \begin{cases} \frac{1}{2g} \left[1 + g^2 - \left(\frac{1 - g^2}{1 + g(2\xi - 1)} \right)^2 \right] & \text{if } g \neq 0 \\ 2\xi - 1 & \text{if } g = 0. \end{cases} \quad (39)$$

The azimuthal angle after scattering is distributed uniformly on $[0, 2\pi)$.

5.4 Reflection/refraction

When a ray crosses a voxel wall or PLIC plane where the refractive indices differ across the interface, the ray splits into a transmitted and a reflected ray and the Fresnel coefficients for unpolarized light compute the energy carried by each of the rays. The Fresnel coefficients describe the amplitude and phase shifts of reflected and refracted components due to an electromagnetic wave

incident on a homogenous, planar, non-magnetic interface [15]. The Fresnel coefficients are useful for ray propagation calculations because of the localized and well-defined material and interface conditions in the vicinity of the ray location. SESE considers only the energy transport across the interface and discards phase shift information. Snell's law modifies the direction of the transmitted ray, and specular reflection modifies the direction of the reflected ray, both computed with reference to the orientation of the local PLIC plane.

SESE currently assumes that sources of radiation are unpolarized, so the average of the Fresnel coefficients for s - and p -polarized light determines the amount of energy carried by each ray after a surface interaction. The fraction of the initial ray weight assigned to the reflected ray is

$$R = \frac{R_s + R_p}{2} \quad (40a)$$

$$R_s = \left| \frac{n_1 \cos \theta_1 - n_2 \cos \theta_2}{n_1 \cos \theta_1 + n_2 \cos \theta_2} \right|^2 \quad (40b)$$

$$R_p = \left| \frac{n_1 \cos \theta_2 - n_2 \cos \theta_1}{n_1 \cos \theta_2 + n_2 \cos \theta_1} \right|^2 \quad (40c)$$

where θ_1 , θ_2 are the angles of incidence and refraction, respectively, and n_1 , n_2 are the indices of refraction of the material on the incident and the transmitted sides of the interface, respectively, and $\hat{n}$ is the interface normal. If the incident ray direction unit vector is $\hat{i}$ and the surface normal vector is $\hat{n}$, then the reflected ray's direction unit vector is

$$\hat{r} = \hat{i} - 2(\hat{i} \cdot \hat{n}) \hat{n} \quad (41)$$

Conservation of energy finds the fraction of the initial ray weight assigned to the transmitted ray

$$T = 1 - R \quad (42)$$

The transmitted ray direction is determined such that the incident $\hat{i}$ and transmitted $\hat{t}$ ray direction unit vectors are coplanar with the normal vector and θ_1 , θ_2 satisfy Snell's law at the interface

$$\sin \theta_2 = \frac{n_1}{n_2} \sin \theta_1 \quad (43)$$

The transmitted ray direction is a weighted sum of the surface normal vector and the incident ray normal vector

$$\hat{i} = \frac{n_1}{n_2} \hat{i} + \left(\frac{n_1}{n_2} (\hat{i} \cdot \hat{n}) - \sqrt{1 - \left(\frac{n_1}{n_2} \right)^2 (1 - (\hat{i} \cdot \hat{n})^2)} \right) \hat{n} \quad (44)$$

In certain cases, the transmitted ray direction can be imaginary. This occurs when

$$\frac{n_1}{n_2} \sin \theta_1 > 1 \quad (45)$$

and corresponds to the ray experiencing total internal reflection.

Since in general the piecewise planar interface has small gaps and/or discontinuities at the voxel walls, rays may propagate to a voxel face at a discontinuity where there is no interface plane. This case uses the voxel wall normal in lieu of the PLIC plane. Hence, for the radiative transport algorithm, the voxel walls serve as part of the interface at which refraction and reflection may occur.

5.5 Smart Dose Recalculation

During the course of a simulation, the changing of dynamic optical properties (refractive index, scattering coefficient, absorption coefficient), which vary with temperature and/or chemical damage, may require recomputation of the distribution of absorbed radiation (the dose distribution). If the optical constants of the materials in the object and the location and characteristics of the radiation sources are constant over the course of a computational timestep, it is possible to reuse the previously computed dose distribution in order to prevent unnecessary expenditure of computational effort. Otherwise, changes in the following properties in the domain may require recomputation of the dose distribution:

- Absorption coefficient α
- Scattering coefficient σ
- Scattering anisotropy parameter g
- Refractive index n
- Volume fraction f
- Source position, size, power, wavelength, orientation or other characteristics

SESE lets the user specify when to reuse the radiative dose. If, during a time step, any radiation source's position, size, power, wavelength, orientation, or other characteristics have changed, a new Monte Carlo ray tracing step will always be triggered.

Otherwise, detecting changes in the remaining optical properties sufficient to warrant dose recomputation requires a method that keeps track of the global maximum change in any optical coefficient throughout the domain Ω since the last dose computation, and triggers a recomputation when this change exceeds a given threshold.

Suppose the last dose distribution was computed at time t_0 . Consider a change in the spatially- and temporally-varying absorption coefficient $\alpha(x, t)$, where there is an explicit dependence of α on x and t via the temperature $T(x, t)$ and damage $D(x, t)$ fields. Ideally, a new Monte Carlo computation would be triggered at $t = t_k$ when the change of α in some voxel has exceeded a user-defined threshold ε ,

$$M_1 \equiv \max_{x \in \Omega} \{|\alpha(x, t_k) - \alpha(x, t_0)|\} \geq \varepsilon \quad (46)$$

However, this would require storage of the entire field of $\alpha(x, t_0)$, or at least $T(x, t_0)$ and $D(x, t_0)$, for times $t_0 \leq t \leq t_k$ in order to perform this comparison after each time step. Instead, note that by the triangle inequality,

$$|\alpha(x, t_k) - \alpha(x, t_0)| \leq |\alpha(x, t_k) - \alpha(x, t_{k-1})| + \dots + |\alpha(x, t_1) - \alpha(x, t_0)| \quad (47)$$

Hence,

$$\max_{x \in \Omega} \{|\alpha(x, t_k) - \alpha(x, t_0)|\} \leq \max_{x \in \Omega} \left\{ \sum_{i=1}^k |\alpha(x, t_i) - \alpha(x, t_{i-1})| \right\} \quad (48a)$$

$$\leq \sum_{i=1}^k \max_{x \in \Omega} \{|\alpha(x, t_i) - \alpha(x, t_{i-1})|\} \equiv M_2 \quad (48b)$$

Therefore, if $M_1 \geq \varepsilon$ then also $M_2 \geq M_1 \geq \varepsilon$. This shows that by simply accumulating the maximum change of α over the domain after each time step, a criterion for triggering a Monte Carlo step is at least as strict as the original criterion.

If this new criterion is too strict, however, the dose recomputation may be triggered too often, when $M_2 \geq \varepsilon > M_1$. For example, this could occur if the maximum change in α occurs in a different voxel in each time step, or if the change in α is oscillatory in time, since local negative and positive changes in α contribute equally to M_2 but cancel in M_1 . However, in typical scenarios of rapid heating dominated by a static radiative source term, with monotonic dependence of optical coefficients on temperature and/or damage, the temperature change in each voxel is indeed monotonic with time and the maximum change occurs in the same region at every time step, implying that M_1 is well-approximated by M_2 .

Note that rather than accumulating the absolute change $\Delta\alpha$ at each time step, the relative change $\Delta\alpha/\alpha$ is accumulated instead, which allows ε to be specified as a dimensionless threshold for relative change. As long as $\Delta\alpha/\alpha$ is small, to first order this is equivalent to accumulating the absolute change $\Delta\alpha$ with the dimensional threshold $\varepsilon' = \alpha\varepsilon$. This is the approach implemented in SESE.

This method for triggering dose recomputation has the advantage that the additional memory requirements are negligible (storing only a single extra number M_2), although it does require an extra calculation of the optical properties over the entire active domain after each time step, which will typically consume fewer computational cycles than performing another Monte Carlo simulation.

A note of caution – since there is sampling noise present in the radiative source term calculated in the Monte Carlo step, re-using the same source term across multiple time-steps causes the errors in the calculated temperature due to statistical sampling variations in the source term to grow linearly in time. Therefore, ensure that the computed source term is of sufficient quality, which may require longer computation in the initial Monte Carlo step.

5.6 Dose Uncertainty Estimation

Since SESE's Monte Carlo radiative transport method is a statistical method, it is important to quantify and limit the uncertainty in the dose distribution resulting from sampling error. To achieve this, SESE divides each Monte Carlo radiative transport simulation into several independent realizations of the radiative dose and analyzes the results of each to determine the overall uncertainty in the total dose distribution. SESE continues to run additional realizations until the estimated uncertainty falls below a user-defined threshold.

5.6.1 Partition scheme.

In order to calculate the statistical variance in the dose distribution found through the Monte Carlo method, a number of realizations called 'partitions' divides the entire radiative dose computation. Each partition propagates a fixed number of rays and tallies the resulting dose distribution in a region of interest. Each partition represents a statistically-independent sample of the entire dose distribution and enables run-time statistical analysis of the Monte Carlo results. The simulation run proceeds, computing additional partitions and updating the statistics of the ensemble of partitions.

If a sufficient number of rays are sampled in each partition, the dose in each voxel x in the region of interest can be considered as an independent sample from a Gaussian distribution with some mean dose μ_x and variance σ_x^2 , quantities which are estimated from the ensemble of all partitions. This variance estimates the uncertainty in the ensemble average dose.

5.6.2 Region of interest selection.

To ensure sampling a sufficient number of rays in each partition to produce a reasonable uncertainty estimate, the set of the M highest-energy voxels in the dose distribution in the first partition defines a region of interest (ROI). The ROI size M is defined by the user; by default, $M = 100$. During the first partition, rays propagate until each of the M highest-energy voxels have received energy deposited by at least 10 different rays. Meeting this criterion completes the first partition. This determines the fixed number of rays to propagate in each partition. Completing the first partition

fixes the voxels comprising the ROI for the remaining partitions.

5.6.3 Overall uncertainty metric.

To compute an overall uncertainty estimate for the dose distribution, the sample mean μ_x of the energy $e_{j,x}$ deposited in each voxel x in the ROI is computed, across the ensemble of partitions j up to the current partition N :

$$\mu_x = \frac{1}{N} \sum_{j=1}^N e_{j,x} \quad (49)$$

Then the sample standard deviation of the deposited energy across the ensemble of partitions for each voxel x :

$$\sigma_x = \left[\frac{1}{N-1} \sum_{j=1}^N (e_{j,x} - \mu_x)^2 \right]^{1/2} \quad (50)$$

Accordingly, the estimated uncertainty s_x of the mean deposited energy μ_x in each voxel for a partition is, by the central limit theorem,

$$s_x = \frac{\sigma_x}{\sqrt{N}} \quad (51)$$

5.6.4 Online variance calculation.

A notable issue with the method to determine uncertainty of the dose distribution as described above is that it requires the full ensemble of deposited energies $e_{j,x}$, i.e. an additional value for each voxel in the ROI for each partition. This means the total number of values required to compute the error estimate is MN , where M is the number of voxels in the region of interest and N is the number of partitions.

To keep track of the required information without requiring a large amount of additional memory, an ‘online’ algorithm [16] for tracking the sample means and sample standard deviations has been implemented in SESE, requiring only $2M$ additional values in order to store the information needed to compute the estimated uncertainty s_x during the simulation. The online equation for the updated sample mean energy $\mu_{x,n}$ in voxel x for a new partition number n is

$$\mu_{x,n} = \mu_{x,n-1} + \frac{e_{x,n} - \mu_{x,n-1}}{n} \quad (52)$$

Similarly, the updated sample standard deviation $\sigma_{x,n}$ for voxel x for a new partition n is found with the expression

$$\sigma_{x,n}^2 = \frac{1}{n-1} \left[(n-2)\sigma_{x,n-1}^2 + \left(\frac{n-1}{n} \right) (e_{x,n} - \mu_{x,n-1})^2 \right] \quad (53)$$

5.6.5 Automatic termination.

To stop the ray tracing step based on the uncertainty of the dose distribution, a single number is required that measures the overall uncertainty associated with all of the voxels within the ROI. An energy-weighted relative standard error summed over the set of voxels in the region of interest finds this value:

$$U = \frac{\sum_x \frac{s_x}{\mu_x} \mu_x}{\sum_x \mu_x} = \frac{\sum_x s_x}{\sum_x \mu_x} \quad (54)$$

This estimate is dimensionless and weights voxels that contribute the most energy to the dose distribution more than those that contribute less energy. Empty regions do not contribute. The radiative transport portion of a simulation run accumulates and analyzes partitions until the value of U is less than the user-defined uncertainty tolerance parameter. To ensure the uncertainty is not erroneously underestimated, a minimum of ten partitions are required before the simulation may terminate. Figure 6 shows a plot of the convergence of the overall dose uncertainty parameter U , as a function of the number of partitions N executed during a typical radiative dose computation step. In this example, the ROI size was set to the default $M = 100$, and the dose uncertainty tolerance (to end the Monte Carlo radiative transport) was set to 0.01.

5.6.6 Parallel implementation.

In a parallel computing environment consisting of nodes $r = 1, \dots, N$, the root node ($r = 1$) first performs a short initial ray trace step to determine the location of the top M highest-energy voxels. These M voxels become the ROI, and broadcasting their positions to the other nodes, ensures all nodes share the same ROI.

Afterward, each node independently traces rays, generating a number of partitions n and independently calculating the per-voxel means $\mu_{x,n,r}$ and variances $\sigma_{x,n,r}^2$ for each voxel x in the ROI. The nodes communicate these individual means and variances to the root node $r = 1$, which combines the individual per-voxel means and variances over n partitions from the N nodes into the global per-voxel means $\mu_{x,n}$ and variances $\sigma_{x,n}^2$ over $N \cdot n$ partitions. The root node uses these global per-voxel means and variances to calculate the overall global uncertainty U and decides whether the uncertainty threshold has been reached, in which case it informs the other nodes to terminate their respective radiative dose computation steps.

After the global uncertainty U falls below the user-defined threshold, the average of the dose distributions calculated independently by each of the N nodes forms the final result and broadcast

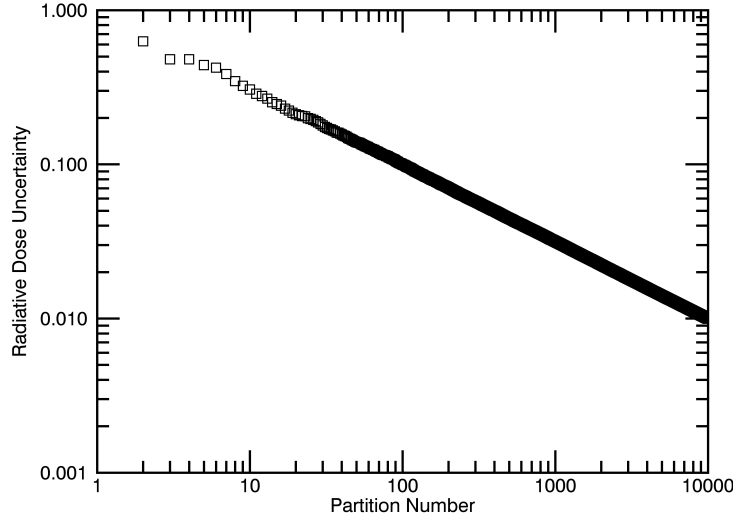


Figure 6. Convergence of the overall dose uncertainty U , as computed by Equation 54, as more rays are accumulated during the Monte Carlo ray tracing step. For this graph, the uncertainty U is computed in the $M=100$ voxels with the highest dose. Rays are accumulated, the uncertainty is periodically measured after each partition, and this process continues until the desired dose uncertainty, here 0.01, is obtained

to all nodes. In this way, all $N \cdot n$ partitions contribute to the final dose distribution.

6.0 THERMAL MODEL

6.1 Theory and Discretization

SESE employs a Gauss-Seidel red-black successive over-relaxation method (see e.g. [1]) to solve the heat equation within the three-dimensional computational domain. A detailed derivation of the method follows.

The heat equation for the time evolution of temperature u as a function of position within the domain, including a perfusive term and the radiative dose distribution Q as a heat source, is

$$\rho c \frac{\partial u}{\partial t} = \nabla \cdot (\kappa \nabla u) + \omega \rho_b c_b (u_b - u) + Q \quad (55)$$

where ρ is density, c is specific heat, κ is thermal conductivity, ω is the perfusion coefficient, ρ_b , c_b , and u_b are the density, specific heat, and temperature, respectively, of the perfusive medium. The material properties are themselves functions of the material ID $M(\mathbf{x})$, and are thus functions of position, temperature $u(\mathbf{x}, t)$ and damage $\Omega(\mathbf{x}, t)$. Material properties have only a weak dependence on u (and thus a weak implicit dependence on time).

To derive a finite-volume discrete approximation, SESE integrates the heat equation over a control

volume v consisting of the region occupied by material within a single voxel with the integer coordinates (i, j, k) :

$$\int_v \rho c \frac{\partial u}{\partial t} dV = \int_v \nabla \cdot (\kappa \nabla u) dV + \int_v \omega \rho_b c_b (u_b - u) dV + \int_v Q dV \quad (56a)$$

$$= \int_{\partial v} \kappa \nabla u \cdot \mathbf{n} dA + \int_v \omega \rho_b c_b (u_b - u) dV + \int_v Q dV \quad (56b)$$

where the divergence theorem has been used to turn the diffusion term into a sum of fluxes over the control volume boundary ∂v . If the control volume v is an interior voxel, the boundary ∂v consists of the six voxel faces shared with the neighboring voxels. If v is a surface voxel, the PLIC plane forms one of the boundary faces, which truncates the faces that the voxel shares with neighboring voxels.

SESE discretizes the temperature and source fields on the computational mesh, defined as the average temperature and dose over the control volume (a single voxel):

$$\bar{u} \equiv \int_v u dV / \int_v dV \quad (57a)$$

$$\bar{Q} \equiv \int_v Q dV / \int_v dV \quad (57b)$$

Hereafter, the overline, which indicates the voxel average, is omitted for notational simplicity and all spatial quantities are understood to be discretized on the computational grid. The volume of material contained within the control volume is simply

$$V \equiv \int_v dV = f h_x h_y h_z \quad (58)$$

where f is the voxel's volume fraction, equal to one for interior voxels and less than or equal to one for surface voxels.

Quantities $a_{i\pm 1/2}$ are defined to be the areas connecting the voxel at coordinates i, j, k to its neighbors in the $\pm x$ directions via the respective voxel faces; the value of this quantity is $h_y h_z$ for interior faces, and is some fraction of $h_y h_z$ for faces that are intersected by the object surface. Similarly, $a_{j\pm 1/2}$ and $a_{k\pm 1/2}$ are defined as the areas for the faces of voxel i, j, k shared by neighbors in the $\pm y$ and $\pm z$ directions, respectively. Finally, for surface voxels with an exposed PLIC surface area a_{surf} , the outward heat flux q_{surf} flows; for interior voxels, $a_{\text{surf}} = 0$ (see Figure 7).

Now, the finite-volume approximation of the heat equation for voxel i, j, k (Equation 56a) becomes:

$$\rho c \frac{\partial u}{\partial t} V = \left[\kappa \frac{\partial u}{\partial x} \right]_{i+1/2} a_{i+1/2} - \left[\kappa \frac{\partial u}{\partial x} \right]_{i-1/2} a_{i-1/2} \quad (59a)$$

$$+ \left[\kappa \frac{\partial u}{\partial x} \right]_{j+1/2} a_{j+1/2} - \left[\kappa \frac{\partial u}{\partial x} \right]_{j-1/2} a_{j-1/2} \quad (59b)$$

$$+ \left[\kappa \frac{\partial u}{\partial x} \right]_{k+1/2} a_{k+1/2} - \left[\kappa \frac{\partial u}{\partial x} \right]_{k-1/2} a_{k-1/2} \quad (59c)$$

$$- q_{\text{surf}} a_{\text{surf}} \quad (59d)$$

$$+ \omega \rho_b c_b V (u_b - u) \quad (59e)$$

$$+ QV \quad (59f)$$

Since the PLIC plane surface interface reconstruction may be discontinuous, the area of the voxel wall adjoining the PLIC plane in voxel i may be different from that which adjoins the PLIC plane in voxel $i + 1$. To reconcile this difference and uniquely define the area available to conduct heat through the voxel wall, the areas $a_{i\pm 1/2}$, $a_{j\pm 1/2}$, $a_{k\pm 1/2}$ are set to be the geometric average of the different areas adjoining the PLIC planes on both sides of the voxel wall; e.g.

$$a_{i-1/2} = \sqrt{(a_{i,j,k})_{i-} (a_{i-1,j,k})_{i+}} \quad (60a)$$

$$a_{i+1/2} = \sqrt{(a_{i,j,k})_{i+} (a_{i+1,j,k})_{i-}} \quad (60b)$$

where $(a_{i,j,k})_{i\pm}$ is the area of voxel i, j, k 's face in the $+x$ and $-x$ direction, respectively. The surface area a_{surf} is the sum of the PLIC plane surface area and the projected areas of outward-facing voxel walls in the direction of the normal vector.

$$a_{\text{surf}} = a_{\text{PLIC}} + \sum_{\tilde{a}} \max(\tilde{a} \cdot \hat{n}) \quad (61)$$

where $\tilde{a}$ denotes the outward area vector of each exposed voxel wall (which will always be parallel or anti-parallel to one of the Cartesian grid directions), and $\hat{n}$ is the normal unit vector of the PLIC plane. The magnitude of the outward area vector is the positive difference between the portion of the voxel wall under the local PLIC plane and that lying under the neighboring PLIC plane. This positioning is more clearly shown in Figure 7. For example, in the positive x direction,

$$\tilde{a}_{i+} = |(a_{i,j,k})_{i+} - (a_{i+1,j,k})_{i-}| \hat{x} \quad (62)$$

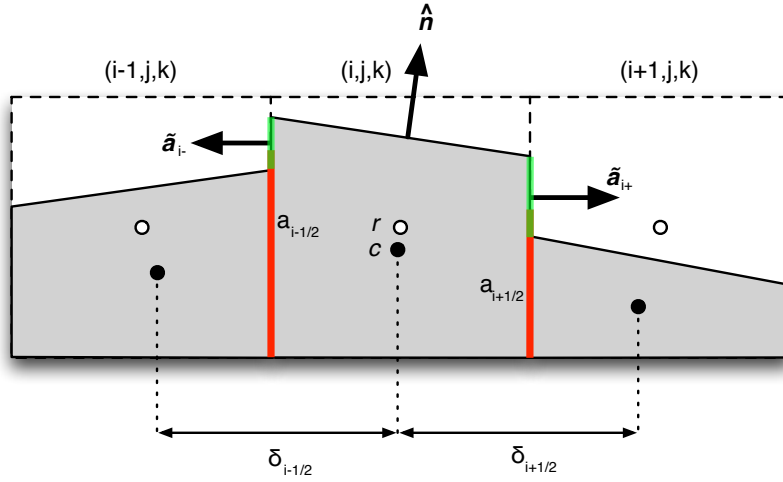


Figure 7. Geometric quantities in the finite volume discretization for voxels at the surface of the object

SESE discretizes the time derivative by the backward time difference

$$\frac{\partial u}{\partial t} \approx \frac{u - u_{\text{prev}}}{\Delta t} \quad (63)$$

The spatial derivatives in the diffusion term are discretized using centered differences at each voxel face

$$\pm \left[\kappa \frac{\partial u}{\partial x} \right]_{i \pm 1/2} \approx \kappa_{i \pm 1/2} \frac{u_{i \pm 1} - u}{\delta_{i \pm 1/2}} \quad (64)$$

where $\delta_{i \pm 1/2}$ is the distance between the centroid of voxel i, j, k and that of its neighbor in the $\pm x$ direction, projected onto the x -axis. Fluxes in the other directions have similar definitions. In interior voxels, the centroids coincide with the voxel centers and the distance is simply $\delta_{i \pm 1/2} = h_x$, but in surface voxels one must calculate the location of the voxel centroids. Currently, the location of the voxel centroid c is approximated by the location of the intersection of a line extending from the voxel center r inward along $-\hat{n}$ to the orthogonal plane that divides the voxel into two sections of equal volume $f/2$:

$$c = r - |\alpha(\hat{n}, 1/2) - \alpha(\hat{n}, f/2)| \hat{n} \quad (65)$$

where $\alpha(\hat{n}, f)$ is the PLIC distance for a voxel with normal vector $\hat{n}$ and volume fraction f . The

x -projected distance between the approximate voxel centroid c and that of its neighbor $c_{i\pm 1/2}$ approximates the separation $\delta_{i\pm 1/2}$:

$$\delta_{i\pm 1/2} = \|(c - c_{i\pm 1/2}) \cdot \hat{x}\| \quad (66)$$

and similarly for $\delta_{j\pm 1/2}$ and $\delta_{k\pm 1/2}$. For normal vectors aligned with the principal grid axes, this construction yields the exact location of the voxel centroid, but in other configurations it is an approximation.

The thermal conductivity at the wall is the average of that of the voxel and its neighbor:

$$\kappa_{i\pm 1/2} = \frac{\kappa + \kappa_{i\pm 1}}{2} \quad (67)$$

The remaining thermal conductivities $\kappa_{j\pm 1/2}$, $\kappa_{k\pm 1/2}$ are defined similarly. The surface heat flux q_{surf} is calculated from the user-selected Neumann boundary conditions (convective, evaporative, radiative, or insulating) applied at the surface.

Dividing Equation 59 by V and gathering the terms multiplying u ,

$$\begin{aligned} \left(\frac{\rho c}{\Delta t}\right) (u - u_{\text{prev}}) &= - \left(\frac{\kappa_{i+1/2} a_{i+1/2}}{V \delta_{i+1/2}} - \frac{\kappa_{i-1/2} a_{i-1/2}}{V \delta_{i-1/2}} + \dots \right) u \\ &+ \left(\frac{\kappa_{i+1/2} a_{i+1/2}}{V \delta_{i+1/2}} u_{i+1} - \frac{\kappa_{i-1/2} a_{i-1/2}}{V \delta_{i-1/2}} u_{i-1} + \dots \right) \\ &- (\omega \rho_b c_b) u \\ &- q_{\text{surf}} \frac{a_{\text{surf}}}{V} + \omega \rho_b c_b u_b + Q. \end{aligned} \quad (68)$$

Defining the quantities

$$\gamma_{i\pm 1/2} = \frac{\kappa_{i\pm 1/2} a_{i\pm 1/2}}{V \delta_{i\pm 1/2}} \quad (69)$$

with similar definitions for $\gamma_{j\pm 1/2}$, $\gamma_{k\pm 1/2}$, and further gathering all the terms multiplying u to one side,

$$\begin{aligned} \left\{ 1 + \frac{\Delta t}{\rho c} \left((\gamma_{i+1/2} - \gamma_{i-1/2} + \dots) + \omega \rho_b c_b \right) \right\} u &= u_{\text{prev}} \\ + \frac{\Delta t}{\rho c} \left\{ \left(\gamma_{i+1/2} u_{i+1/2} - \gamma_{i-1/2} u_{i-1/2} + \dots \right) - q_{\text{surf}} \frac{a_{\text{surf}}}{V} + \omega \rho_b c_b u_b + Q \right\} \end{aligned} \quad (70)$$

Letting

$$\alpha = \frac{\Delta t}{\rho c} (\gamma_{i+1/2} - \gamma_{i-1/2} + \dots + \omega_b \rho_b c_b), \text{ and} \quad (71a)$$

$$\beta = \frac{\Delta t}{\rho c} \left\{ (\gamma_{i+1/2} u_{i+1/2} - \gamma_{i-1/2} u_{i-1/2} + \dots) - q_{\text{surf}} \frac{a_{\text{surf}}}{V} + \omega_b \rho_b c_b u_b + Q \right\} \quad (71b)$$

the fully discretized heat equation becomes

$$(1 + \alpha)u = u_{\text{prev}} + \beta \quad (72)$$

where recall that u_{prev} is the temperature from the previous time step. Recognizing that material characteristics and the geometry of the computational mesh itself may be weak functions of time, this equation is now written in a form that allows an iterative method to step the temperature field forward in time. The $(m+1)^{\text{th}}$ iterate of the temperature is computed using

$$u^{(m+1)} \leftarrow \frac{1}{1 + \alpha^{(m)}} \left(u^{(m)} + \beta^{(m)} \right) \quad (73)$$

where the superscript (m) denotes the value of the quantity in the m^{th} iteration. An iteration consists of applying this updating rule once in every non-empty voxel of the domain. The order of updating voxels affects the speed of convergence (Jacobi vs. Gauss-Seidel). Note that the rule for updating a given voxel in 3D Cartesian mesh depends only on the temperatures in the voxel's six nearest neighbors. This temperature relationship suggests a red-black updating scheme, in which voxels are 'colored' in a three-dimensional checkerboard pattern according to their parity

$$i + j + k \pmod{2} \quad (74)$$

Each iteration is performed by updating every even-parity ('black') voxel in the domain, which depend only on their adjacent odd-parity ('red') neighbors, followed by updating all the odd-parity ('red') voxels, which in turn only depend on their 'black' neighbors. This alternating red-black updating scheme exhibits superior convergence (see e.g. [1], [17]) compared to simple Gauss-Seidel iteration.

6.2 Domain Decomposition

Due to its high degree of locality, the iterative successive over-relaxation (SOR) method of Equation 73 lends itself directly to implementation as a coarse-grained parallel iterative algorithm. In an environment consisting of N parallel MPI processing nodes, the computational mesh is decomposed into N slabs along the z axis, each of which is owned by one of the nodes. In addition to the

voxels within its slab, each node also stores an additional ‘halo’ of voxels, which contain a local copy of the nearest neighbors to the given slab, as shown in Figure 8. For the slab decomposition, each node therefore stores two halos: one plane for the boundary to the ‘left’, and one plane for the boundary to the ‘right’. Over the course of each red-black iteration of the heat equation, the temperature in each halo voxel synchronizes with the corresponding voxel in its owner node, which is responsible for the computation of its value. Thus, the halo acts to enforce a Dirichlet-type boundary condition on the slab updated with values from the interior of the adjacent slabs.

Voxels update iteratively according to a red-black SOR scheme, as discussed previously. Two passes through each slab in each iteration in each MPI node update all black voxels, then all red voxels. Between colors, the halo voxels just updated synchronize to neighboring nodes using inter-node communication via MPI.

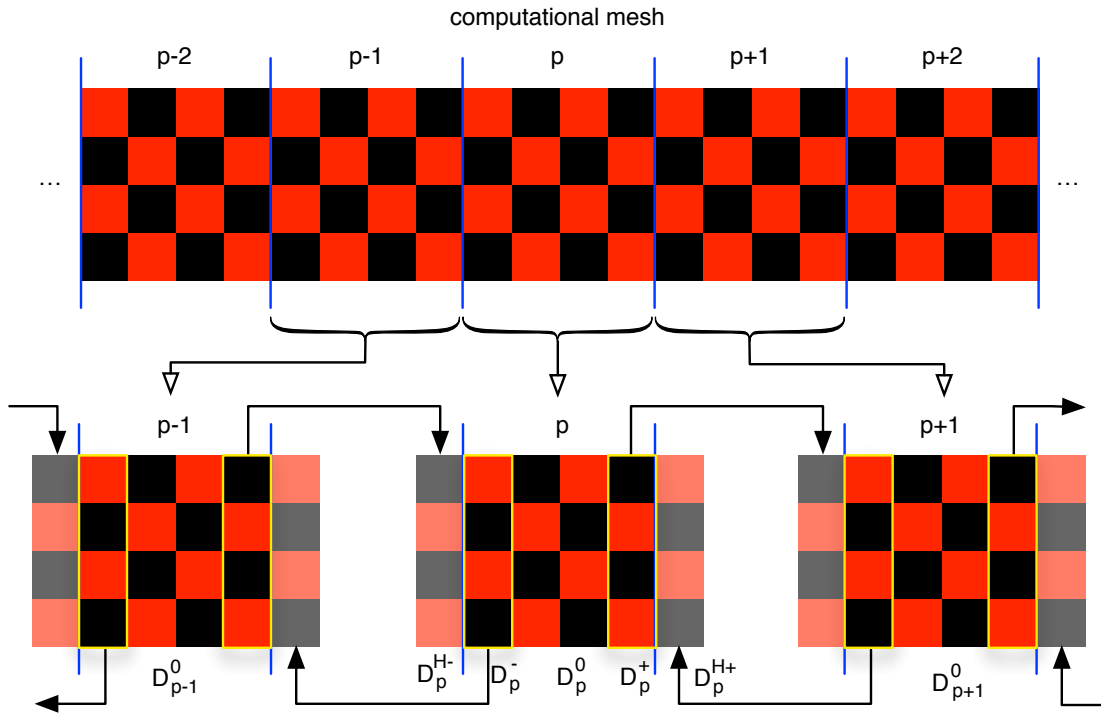


Figure 8. Decomposition of the computational domain into ‘slabs’, and MPI communications between slabs

Each slab is the union of its red and black parts:

$$\begin{aligned}
 D^- &= R^- \cup B^- && \text{(left boundary)} \\
 D^+ &= R^+ \cup B^+ && \text{(right boundary)} \\
 D^{H-} &= R^{H-} \cup B^{H-} && \text{(left halo)} \\
 D^{H+} &= R^{H+} \cup B^{H+} && \text{(right halo)} \\
 D^0 &= R^0 \cup B^0 \cup D^- \cup D^+ && \text{(bulk)}
 \end{aligned}$$

Adopting the convention that in each iteration, black voxels are updated first, then red voxels, the temperature iteration sequence for node p is as follows:

update B_p^0 depends on R_p^0
 transmit B_p^+, B_p^- to $B_{p+1}^{H-}, B_{p-1}^{H+}$
 receive B_p^{H-}, B_p^{H+} from B_{p-1}^+, B_{p+1}^{H-}
 update R_p^0 depends on B_p^0
 transmit R_p^+, R_p^- to $R_{p+1}^{H-}, R_{p-1}^{H+}$
 receive R_p^{H-}, R_p^{H+} from R_{p-1}^+, R_{p+1}^{H-}
 repeat ...

Depending on the eigenvalues of the matrix associated with the iteration in Equation 73, faster convergence may be achieved through the introduction of an over-relaxation parameter η , which mixes the old and new values of each iteration through a linear combination:

$$u^{(m+1)} \leftarrow (1 - \eta) \cdot u^{(m)} + \eta \cdot \frac{1}{1 + \alpha^{(m)}} \left(u^{(m)} + \beta^{(m)} \right) \quad (75)$$

For $\eta = 1$, the algorithm reduces to the previous equation; larger or smaller values of η may be used to either speed convergence (at the expense of stability) or regain stability (at the expense of speed), respectively. This is the algorithm implemented in SESE. Typical values of η between 1.3 to 1.7 provide a good trade off between convergence speed and stability.

6.3 Convergence Detection

Iteration continues until the global maximum magnitude of the temperature change in any voxel per iteration falls below a user-defined threshold. In a parallel computing environment, the maximum iteration change among all MPI nodes is via a call to MPI_Reduce. If the maximum change in the iteration is below the user-defined threshold, the iterative solution has converged and the most recent step is the solution.

6.4 Boundary Conditions

It is possible to define material-, temperature- and damage-dependent thermal boundary conditions at the object surface. SESE currently implements the following thermal heat flux boundary conditions, which the user can select in any combination are:

$$q_{\text{rad}} = \sigma \varepsilon(T, f)(T^4 - T_{\infty}^4) \quad (\text{radiative}) \quad (76a)$$

$$q_{\text{conv}} = h_{\text{conv}}(T, f)(T - T_{\infty}) \quad (\text{convective}) \quad (76b)$$

$$q_{\text{evap}} = h_{\text{evap}}(T, f)(p_{\text{sat}}(T) - \phi \cdot p_{\text{sat}}(T_{\infty})) \quad (\text{evaporative}) \quad (76c)$$

where ε is the surface emissivity, h_{conv} is the convective heat transfer coefficient, h_{evap} is the evaporative heat transfer coefficient and ρ is density, all whose temperature- and damage-dependence are specified on a per-material basis in the material definition file.

The quantity σ is the Stefan-Boltzmann constant, T_{∞} is the temperature of the environment surrounding the object, while ϕ denotes the ambient relative humidity – the simulation configuration file specifies these. The function $p_{\text{sat}}(T)$ is the saturation pressure of water vapor. In SESE, $p_{\text{sat}}(T)$ is computed reliably at any temperature from -50°C to 400°C using either fifth- or sixth-order polynomial fits (depending on the applicable temperature range).

Each material file independently specifies the boundary conditions, allowing application of a variety of different boundary conditions on different areas of the object surface. Simulation configuration file directives globally switch on or off these effects, so that the full heat flux boundary condition at the surface is the sum of the active effects:

$$q_{\text{surf}} = q_{\text{rad}} + q_{\text{conv}} + q_{\text{evap}} \quad (77)$$

By default, switching off all boundary conditions yields the insulating boundary condition.

7.0 ADAPTIVE TIME STEP

SESE's adaptive time step controller limits the error incurred in the temperature field during each time step. First, a 'step doubling' method estimates the error incurred during a time step Δt . Then, a factor q scales the time step to bring the estimated error into line with the user-specified tolerance τ . Too large and error discards the step and another attempt is made with the smaller time step. The user may specify minimum and maximum time step limits as well as minimum and maximum time step scaling factors q to customize the behavior of the time step controller.

7.1 Error Estimator For Heat Equation Solver

The time step must be small enough to represent the time dependent terms in the heat equation (for example, the source term) faithfully. Suppose that our heat solver algorithm has accuracy in time of order p . In particular, the fully implicit heat equation time stepping method used in SESE has order $p = 1$. Now suppose the temperature $u(t)$ at time t has been computed. To advance in time, the numerical solution of the temperature at time $t + \Delta t$ is calculated in two ways. First, via one time step $t \rightarrow t + \Delta t$, computing

$$(t) \rightarrow u(t + \Delta t) \equiv u_1 \quad (78)$$

Next, the algorithm takes two half steps $t \rightarrow t + \Delta t/2 \rightarrow t + \Delta t$, computing

$$u(t) \rightarrow u(t + \Delta t/2) \rightarrow u(t + \Delta t) \equiv u_2 \quad (79)$$

Since it is known that the algorithm has order p , the two procedures incur numerical errors predictably and by comparing the difference between u_1 and u_2 the actual numerical error incurred can be estimated.

Denote the true (unknown) solution by $\hat{u}$. Since the numerical method is of order p , the computed solutions contain local truncation errors of order $p + 1$ and higher:

$$u_1 = \hat{u} + C \cdot \Delta t^{p+1} + O(\Delta t^{p+2}), \text{ and} \quad (80a)$$

$$u_2 = \hat{u} + 2C \cdot (\Delta t/2)^{p+1} + O(\Delta t^{p+2}) = \hat{u} + 2^{-p}C \cdot \Delta t^{p+1} + O(\Delta t^{p+2}) \quad (80b)$$

Subtracting u_2 from u_1 and neglecting higher order terms

$$|u_1 - u_2| = (1 - 2^{-p}) (C \cdot \Delta t^{p+1}) \quad (81)$$

where $|\cdot|$ denotes the L^∞ norm, i.e. the maximum absolute difference within the entire spatial domain. The local truncation error ε_2 in u_2 per time step is therefore, to leading order,

$$\varepsilon_2 \approx 2^{-p} (C \cdot \Delta t^{p+1}) = \frac{1}{2^p - 1} |u_1 - u_2| \quad (82)$$

If a factor q scales the time step, our new solution would have a local error

$$\tilde{\varepsilon}_2 = q^{p+1} \varepsilon_2 \quad (83)$$

We wish to constrain this error to be approximately equal to some small user-defined truncation error tolerance τ :

$$\frac{q^p}{2^p - 1} |u_1 - u_2| \approx \tau \quad (84)$$

therefore, the scaling factor q must be

$$q \approx (2^p - 1)^{1/p} \left(\frac{\tau}{|u_1 - u_2|} \right)^{1/p} \quad (85)$$

The implicit thermal solver implemented in SESE has order $p = 1$, so the scaling factor is simply the ratio of the error tolerance to the solution difference:

$$q = \frac{\tau}{|u_1 - u_2|} \quad (86)$$

If $q < 1/2$, the error is deemed too large and SESE rejects the time step. The calculated factor q scales the new time step, SESE restores the temperature to time t and makes another attempt with the now-smaller time step. Similarly, if the quantity $|u_1 - u_2|$ is small, then $q > 1$, SESE accepts the temperature u_2 as the new temperature, and SESE scales the new time step value by the factor q .

In practice, q should be limited to a maximum value (such as 2) to prevent large jumps in the time step.

7.2 Time Step Controller Algorithm

The timestep controller function is presented here as an algorithm. The variable u^m represents the temperature field at timestep m , with starting value u^0 at time $t = 0$. The operator $A[t]$ advances the time of a temperature field by a time step t , using the RBSOR method. Quantities with a tilde are estimates.

given

user's desired solution time T
user-defined maximum time increment Δt_{max}
user-defined minimum time increment Δt_{min}
error tolerance τ

initialize

start with user-specified maximum timestep: $\Delta t \leftarrow \Delta t_{max}$

$t \leftarrow 0$

$m \leftarrow 1$

$p \leftarrow 1$

repeat

if $\Delta t > T - t$ **then**

limit timestep to finish at time $t = T$: $\Delta t \leftarrow T - t$

```

end if
step  $u^{m-1}$  (via RBSOR) by  $\Delta t$  to get estimate of  $u(t + \Delta t)$ :  $\tilde{u}_1 \leftarrow A[\Delta t]u^{m-1}$ 
initialize guess  $\tilde{u}_2 \leftarrow (u^{m-1} + \tilde{u}_1)/2$  to speed convergence
step  $\tilde{u}_2$  by  $\Delta t/2$ :  $\tilde{u}_2 \leftarrow A[\Delta t/2]u^{m-1}$ 
step by a second  $\Delta t/2$ :  $\tilde{u}_2 \leftarrow A[\Delta t/2]\tilde{u}_2$ 
find the maximum error between the one-step and two-step estimates:
 $r \leftarrow \sup_{x \in \Omega} |\tilde{u}_1(x) - \tilde{u}_2(x)|$ 
compute scale factor:  $q \leftarrow (\tau/r)^{1/p}$ 
if  $q \geq 2$  then
    accept  $\tilde{u}_2$ :  $u^m \leftarrow \tilde{u}_2$ 
    increment time:  $t \leftarrow t + \Delta t$ 
    increase timestep:  $\Delta t \leftarrow 2\Delta t$ 
    if  $\Delta t > \Delta t_{max}$  then
        do not exceed maximum timestep:  $\Delta t \leftarrow \Delta t_{max}$ 
    end if
else
    if  $2 > q \geq 1$  then
        accept  $\tilde{u}_2$ :  $u^m \leftarrow \tilde{u}_2$ 
        increment time:  $t \leftarrow t + \Delta t$ 
        use a more conservative increase of the timestep:  $\Delta t \leftarrow q\Delta t$ 
        if  $\Delta t > \Delta t_{max}$  then
            do not exceed maximum timestep:  $\Delta t \leftarrow \Delta t_{max}$ 
        end if
    else
        reject timestep, try again with smaller increment:  $\Delta t \leftarrow q\Delta t$ 
        if  $\Delta t < \Delta t_{min}$  then
            do not reduce below minimum timestep:  $\Delta t \leftarrow \Delta t_{min}$ 
        end if
    end if
end if
until  $t \geq T$ 
return  $u^m$ 

```

7.3 Practical Considerations For Maximizing Time Step

When running a SESE simulation, the adaptive time step controller tries to balance the size of the simulation time steps with the amount of error that is acceptable to the user. A number of situations may prevent the time step controller from quickly progressing to a large time step. Most of these situations involve initial transients in temperature, heat flux, or radiative dose. Following is a list of situations to try to avoid, if possible, for a scenario:

- Large temperature differences between the object and the ambient (external environment)
- Perfusion pool temperature different than the object temperature
- Large metabolic heat sources or heat sinks in the object

- Thin or "disconnected" portions of the object exposed to radiation source
- Very intense or highly focused radiation sources that produce intense localized heating of the object

8.0 VERIFICATION

8.1 Overview

It is important that SESE simulation results agree with predictions based on analytic models. To this end, methods were devised to verify the correctness of the radiative transport and heat equation solver. SESE ran simulations of physical scenarios that have analytic solutions, and comparisons were made – in all cases excellent agreement was found with theory.

8.2 Radiative Transport Verification

Consider energy deposition due to a point source of radiation contained within an infinite, homogeneous medium with optical absorption and scattering coefficients α and σ , respectively. In this case, the calculation of the energy distribution becomes analytically tractable in two regimes.

Noting that the system has spherical symmetry, the total energy $E(R)$ deposited outside a concentric sphere of radius R due to the point source of flux Φ analytical. For simplicity, let all rays carry an initial energy of 1.

8.2.1 Predominantly absorbing medium ($\alpha \gg \sigma$).

In this case scattering may be neglected completely, since rays will be quickly absorbed before traveling a distance comparable to $1/\sigma$. Consider an ensemble of rays containing total energy E_0 released isotropically from the origin. Since all rays emanate directly outward, losing energy to the material through absorption along their paths, the energy deposited outside the sphere of radius R will follow an exponential decay:

$$E(R) = E_0 e^{-\alpha R} \quad (87)$$

8.2.2 Predominantly scattering medium ($\alpha \ll \sigma$).

If rays are scattered much more strongly than they are absorbed, then each ray's path is a random walk within the homogeneous medium. In this case, the diffusion approximation becomes appropriate. Consider an ensemble of rays containing total energy E_0 released isotropically from the origin; initially, the path length traversed by each ray is $r = 0$. The ray density as a function of time is the fundamental solution to the diffusion equation:

$$\rho(r, t) = \frac{E_0}{(4\pi Dt)^{3/2}} e^{-r^2/4Dt} \quad (88)$$

where, by the Eddington approximation [18], the diffusion coefficient $D = (3(\alpha + \sigma))^{-1}$. Since the energy carried by each ray after traveling a path length t is

$$W(t) = e^{-\alpha t} \quad (89)$$

the rate at which a ray loses energy is dW/dt ; therefore, the rate at which the material gains energy from a ray is $-dW/dt$. By integrating the ray density multiplied by the rate of energy deposition over all $t \in [0, \infty)$, the density of energy deposited within the material after all rays have been completely absorbed is:

$$e(r) = - \int_0^\infty \rho(r, t) \frac{dW}{dt} dt = E_0 \mu^2 \frac{e^{-\mu r}}{4\pi r} \quad (90)$$

where $\mu = \sqrt{3\alpha(\alpha + \sigma)}$. Finally, integrating over the region contained outside the sphere of radius R ,

$$E(R) = \int_R^\infty e(r) \cdot 4\pi r^2 dr = E_0(1 + \mu R)e^{-\mu R} \quad (91)$$

To test radiation transport in the strong absorption and strong scattering regimes, we considered a cube with sides of length 10 m (100 voxels with a voxel spacing of 0.1 m). The material within the cube had absorption and scattering coefficients chosen to observe the decay of $E(R)$ in sufficient detail, yet a negligible portion of the ray energy would escape the block (less than 0.1%). Rays originated from an isotropic point source located at the center of the block. In these tests, $N = 1$ million rays were simulated.

To test the highly-absorbing regime $\alpha \gg \sigma$, the material in the block was assigned optical coefficients $\alpha = 1.0/\text{m}$, $\sigma = 0.01/\text{m}$. To test the highly-scattering regime $\alpha \ll \sigma$, the material was assigned optical coefficients $\alpha = 0.1/\text{m}$, $\sigma = 10.0/\text{m}$, and anisotropy parameter $g = 0$.

In both cases, SESE ran with a single radiative step, and IDL script computed the radial energy distribution from the output temperature field. The simulation results are in excellent agreement with the analytical predictions, and are shown in Figure 9 and Figure 10. In both cases the simulated $E(R)$ fits the analytic prediction very closely, except for one of the first data points for small R . This discrepancy is not an error in the simulation, but is actually an artifact due to measuring a spherical volume in a small number of cubic voxels.

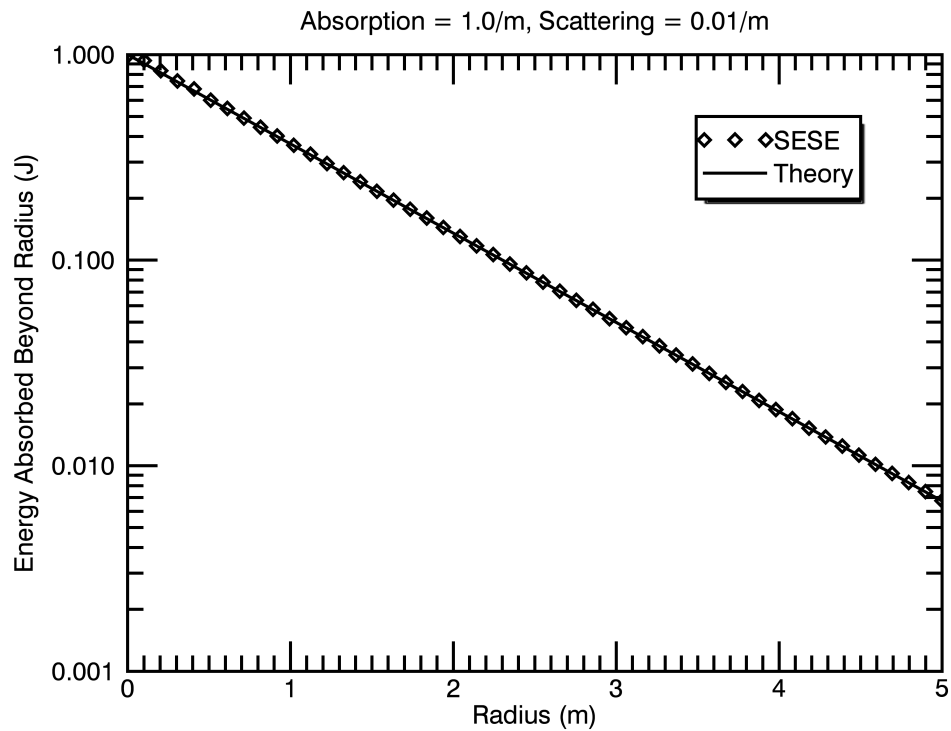


Figure 9. The deposited radial energy distribution $E(R)$ computed by SESE (diamonds) versus that expected from theory (solid line), for a highly-absorbing material. The initial energy carried by each ray is set to $1/N$ for simplicity so that $E(0) = 1$. The total number of rays $N = 10^6$

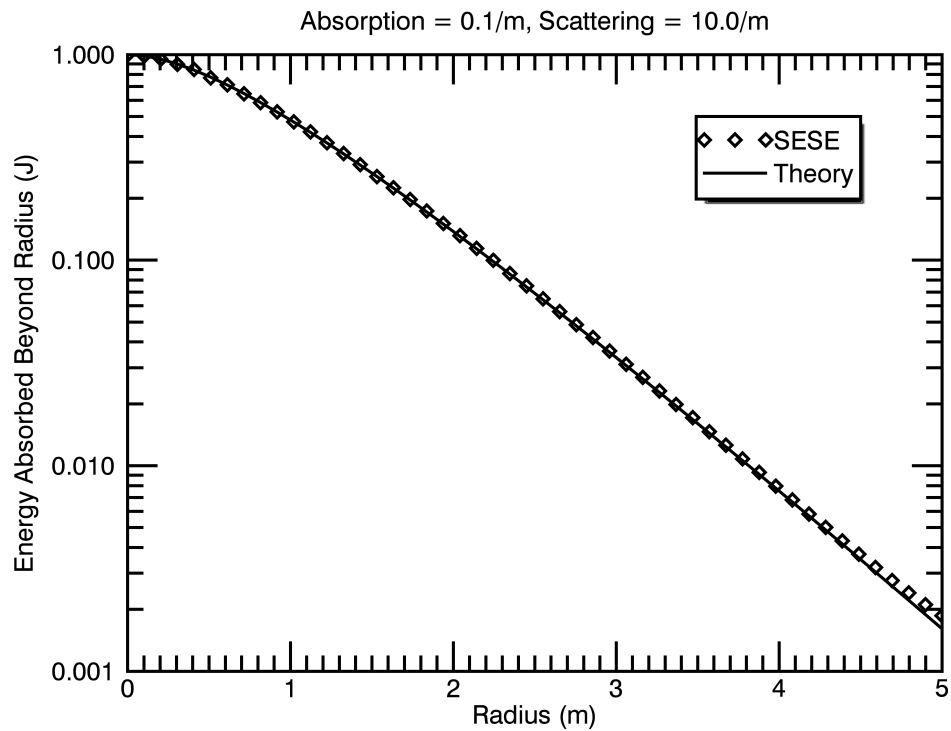


Figure 10. The deposited radial energy distribution $E(R)$ computed by SESE (diamonds) versus that expected from theory (solid line), for a highly-scattering material. The initial energy carried by each ray is set to $1/N$ for simplicity so that $E(0) = 1$. The total number of rays $N = 10^6$

8.3 Thermal Diffusion Verification

Verification of thermal diffusion calculations can be difficult in a code like SESE, where transient behavior typically dominates the solution of the heat equation, and theoretical computation of the solution is difficult, even for simple geometries. These facts motivated the decision to investigate steady-state thermal diffusion due to a heat source-sink "dipole" in a homogeneous medium, a situation with a Green's function solution and bounded steady-state behavior. Under these conditions, the heat equation for the temperature field u simplifies to:

$$\kappa \nabla^2 u + S = 0 \quad (92)$$

where S is a heat source term. For a point source of heat of strength S_0 positioned at the origin,

$$\nabla^2 u = -\frac{S_0}{\kappa} \delta(x) \quad (93)$$

From symmetry, a reasonable solution would be to assume that the temperature is only a function of distance from the origin, *i.e.* $u = u(r)$. Now, integrating the previous equation over all space,

$$\int_V \nabla^2 u(r) dV = -\frac{S_0}{\kappa} \int_V \delta(r) dV \quad (94)$$

allows application of Green's theorem to convert the left-hand volume integral to an integral over a surface that encloses the origin,

$$\int_S \nabla u(r) \cdot \hat{n} dS = -\frac{S_0}{\kappa} \quad (95)$$

If the integral is over a spherical surface of radius R , the temperature gradient is constant over the surface, and parallel to the surface normal vector, so is brought outside the integral. Thus,

$$\int_S \nabla u(r) \cdot \hat{n} dS = u'(R) \int_S dS = 4\pi R^2 u'(R) \quad (96)$$

Now the derivative of the radial temperature, at any value of R ,

$$u'(R) = -\frac{S_0}{4\pi\kappa R^2} \quad (97)$$

can be integrated to find the Green's function solution for the temperature as a function of distance from a point heat source

$$u(r) = \frac{S_0}{4\pi\kappa r} \quad (98)$$

In SESE, since the computational domain is finite, a single heat source would cause the temperature to increase over time without bound, because there is no "infinite boundary condition" implemented in the code. Instead, for this verification test, we use a point source and a point heat sink of equal strength, so that the total heat flux into the computational domain was zero. The two sources in the heat "dipole" were configured a set distance along the x-axis in a homogeneous cube of copper with edge dimension of one meter. SESE ran with no optical sources until reaching steady-state conditions. The comparison between the SESE solution and the theoretical Green's function solution is shown in the left side of Figure 11. The right-side of the figure is a temperature contour plot in the x-y plane, clearly showing the line of symmetry bisecting the dipole. The contour plot also shows finite-boundary artifacts toward the edge of the computational domain, with the contour lines becoming normal to the boundary, a result of the domain's insulating boundary conditions.

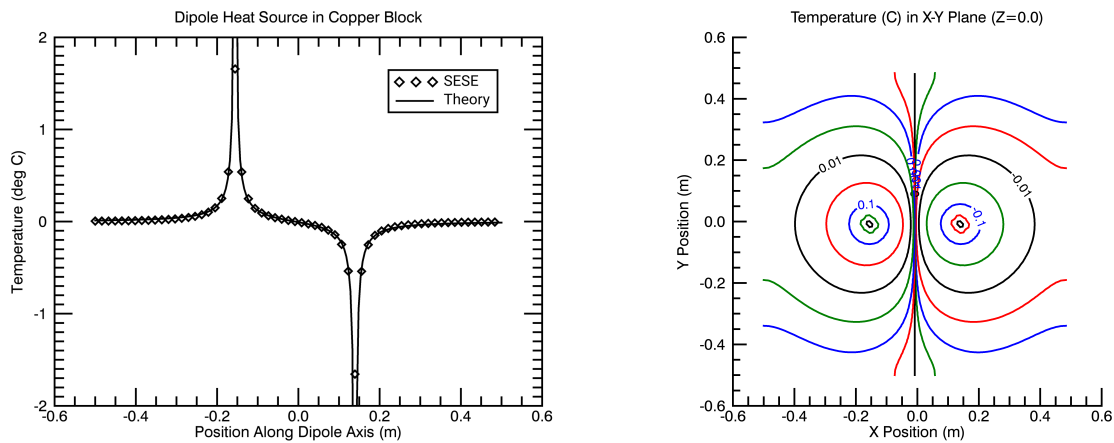


Figure 11. Temperature along x-axis, compared to theory (left) and temperature contour plot in x-y plane (right) for a separated thermal point source and point sink (thermal dipole) in a copper cube after reaching steady state conditions

8.4 Thermal Damage Verification

SESE computed the accumulated Arrhenius damage in a material at constant temperature, for two different temperatures, using reaction values from literature [19]. For this scenario, simulated epidermis material had an Arrhenius prefactor of $1.3\text{E}95 \text{ sec}^{-1}$ and an activation energy of 60.4 kJ/mol, subjected to temperatures of 55°C and 59°C. Figure 12 plots the SESE-calculated and theoretical damage integral vs. time for the two temperatures, showing excellent agreement.

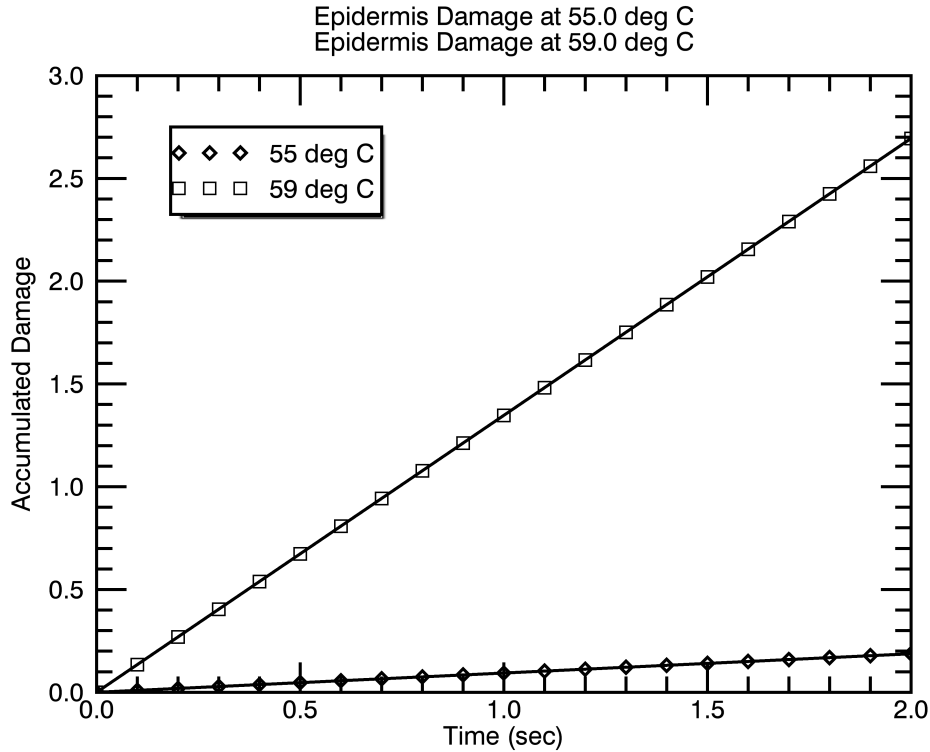


Figure 12. Accumulated thermal damage computed as a function of time for epidermis at 55°C and 59°C (diamonds and squares, respectively), compared with Arrhenius damage integral (solid line)

8.5 Cross-Validation Of SESE Against BTEC

SESE three-dimensional simulation results were compared with the output of the BTEC (Buffington, Thomas, Edwards, and Clark) (version 5.0a2) code. The BTEC code simulates radiation deposition and solution of the heat equation for inhomogeneous layered material compositions in a two-dimensional cylindrical coordinate system (temperature as a function of radius and z-coordinate). In the cross-validation simulation, both SESE (version 1.1) and BTEC modeled a homogeneous block of material with an absorption coefficient of 50 m^{-1} and a scattering coefficient of 10^4 m^{-1} , at a wavelength of $2 \mu\text{m}$. The SESE simulation used a block of dimension $0.1 \times 0.1 \times 0.1 \text{ m}$, and the BTEC simulation used a block with radius 0.05 m and depth 0.1 m . The incident laser beam had a wavelength of $2 \mu\text{m}$, a power of 3.5 W , and a uniform irradiance distribution over a circular (top-hat) cross section of radius 0.02 m . After the two simulations completed, a single plot, Figure 13, combined their respective radiative dose outputs for comparison. Post-run analysis indicates that the integrated RMS difference between the computed doses is less than 1% of the integrated power.

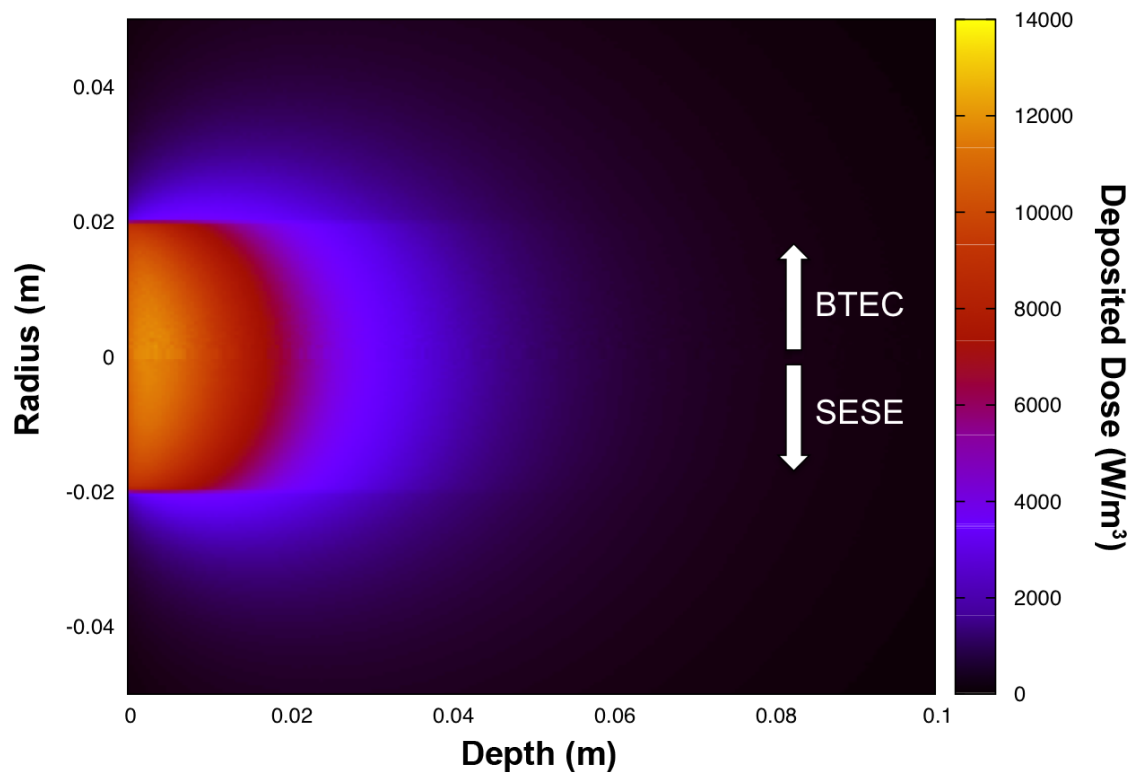


Figure 13. Comparison of the profile of the deposited radiation dose computed with SESE (version 1.1) (lower half of plot) and BTEC (version 5.0a2) (upper half). The computations simulate a circular laser beam incident on an absorbing and scattering material

9.0 REFERENCES

1. C. Zhang, H. Lan, and B. D. Estrade, "Parallel SOR iterative algorithms and performance evaluation on a Linux cluster," *Proceedings by the International Conference on Parallel and Distributed Processing Techniques and Applications*, vol. 1, no. 2, pp. 263–268, Jun. 2005 3, 28, 33.
2. I. Yavneh, "On Red-Black SOR smoothing in Multigrid," *SIAM Journal of Scientific Computing*, vol. 17, pp. 180–192, 1996 3.
3. D. M. Young Jr., "Iterative methods for solving partial difference equations of elliptical type," PhD thesis, Harvard University, May 1950 3.
4. B. J. Parker and D. L. Youngs, "Two and three dimensional Eulerian simulation and fluid flow with material interfaces," UK Atomic Weapons Establishment, Tech. Rep., Jan. 1992 6.
5. R. Scardovelli and S. Zaleski, "Analytical relations connecting linear interfaces and volume fractions in rectangular grids," *Journal of Computational Physics*, vol. 164, pp. 228–237, 2000 6, 7.
6. G. Elpers, "Implementation of the piecewise linear interface calculation (PLIC) boundary representation in SESE," Technical Note TN1064-4, Nanohmics Inc., Jun. 2013 7.
7. A. J. Welch and M. J. van Gemert, Eds., *Optical-Thermal Response of Laser-Irradiated Tissue*, second. Springer, 2011, ISBN: 978-90-481-8831-4 9.
8. A. J. Walker, "New fast method for generating discrete random numbers with arbitrary frequency distributions," *Electronics Letters*, vol. 10, no. 8, pp. 127–128, Apr. 1974, ISSN 0013-5194 12.
9. M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Transactions on Modeling and Computer Simulation*, vol. 8, no. 1, pp. 3–30, Jan. 1998 12.
10. S. Asmussen and P. W. Glynn, *Stochastic Simulation: Algorithms and Analysis*. Springer, 2007, vol. 57, 476 pages, ISBN: 978-0-387-30679-7 12.
11. A. Keller and L. Grüenschloss, "Monte carlo and quasi-monte carlo methods 2010," in, ser. Springer Proceedings in Mathematics and Statistics. Springer, Jul. 2012, vol. 23, ch. 8, pp. 201–213, Parallel Quasi-Monte Carlo Integration by Partitioning Low Discrepancy Sequences 13.
12. J. H. Halton, "Algorithm 247: Radical-inverse quasi-random point sequence," *Communications of the ACM*, vol. 7, no. 12, pp. 701–702, Dec. 1964 13.
13. L. Henyey and J. Greenstein, "Diffuse radiation in the galaxy," *Astrophysical Journal*, vol. 93, pp. 70–83, Jan. 1941 21.
14. S. Prahl, M. Keijer, S. Jacques, and A. Welch, "A Monte Carlo model of light propagation in tissue," *SPIE Institute Series*, vol. IS, no. 5, pp. 102–111, 1989 21.

15. E. Hecht, *Optics*, second. Addison-Wesley, 1987, ISBN: 0805385665 22.
16. P. Pébay, “Formulas for robust, one-pass parallel computation of covariances and arbitrary-order statistical moments,” Sandia National Laboratories, Tech. Rep., Sep. 2008, SAND2008-6212 26.
17. H. Niki, H. Sawami, and T. Ondoh, “A note on a practical technique for solving elliptic equations,” *IMA Journal of Applied Mathematics*, vol. 22, no. 3, pp. 353–359, 1978 33.
18. G. C. Pomraning, “Maximum entropy eddington factors and flux limited diffusion theory,” *Journal of Quantitative Spectroscopy and Radiative Transfer*, vol. 26, no. 5, pp. 385–388, Nov. 1981 41.
19. K. R. Diller and G. A. Klutke, “Accuracy analysis of the Henriques model for predicting thermal burn injury,” *American Society of Mechanical Engineers, Heat Transfer Division*, vol. 268, pp. 117–123, 1993 45.

APPENDIX A - INSTALLATION

A.1 Dependencies

SESE relies on a few external libraries for some special functionality like random number generation and parsing of YAML documents. If SESE runs on an institutional computer setup, ask your administrator to make these libraries available to you. If you plan to run SESE on a workstation, then you will have control over placement of libraries. In a typical UNIX/LINUX environment, the directory `/usr/local` is the place where user-added libraries, headers, and binary programs are stored. For example, `/usr/local/lib` is generally where add-on libraries are installed. Note that to install files in the `/usr/...` tree, you may need to use the `sudo` command and supply an administrator password.

Note that C++ libraries and source code are not guaranteed to be compatible with one another at link time unless they are compiled using the same version of the same compiler. The reason is that there is no standard for name "mangling" (which is the method by which C++ detects overloaded symbols). Thus, it is important that SESE and all of its dependencies are compiled with the same C++ compiler, and that it is specified explicitly at `configure` time. Source code and libraries written in C do not share this problem, and any compiler may be used. SESE was tested extensively using the GNU `gcc` and `g++` compilers. These compilers are recommended for use during the build of SESE and its dependencies.

- **boost (header-only)** — C++ productivity libraries

URL: <http://www.boost.org/>

The Boost library headers are required by `yaml-cpp`. A full build of the Boost libraries is not required. After uncompressing the Boost source tarball, simply copy the headers to your include directory; for example:

```
| tar zxvf boost_1_58_0.tar.gz
| cd boost_1_58_0
| [sudo] cp -R boost /usr/local/include
```

- **yaml-cpp (version 0.5.3)** — a YAML parser and emitter in C++

URL: <https://github.com/jbeder/yaml-cpp/releases/tag/release-0.5.3>

Required to parse SESE's YAML configuration files. To compile `yaml-cpp` from source, you will need to use CMake with the `-DBUILD_SHARED_LIBS=ON` option. As this project is under active development, please check out the newest version available from the github repository.

```
| tar zxvf yaml-cpp-release-0.5.3.tar.gz
| cd yaml-cpp-release-0.5.3
| mkdir build
| cd build
| cmake -DCMAKE_EXE_LINKER_FLAGS="-lboost_regex" -DBUILD_SHARED_LIBS=ON
| -DCMAKE_CXX_COMPILER=g++ -DCMAKE_C_COMPILER=gcc ..
| make
| [sudo] make install
```

- **GNU Scientific Library (GSL)** — a numerical library for C/C++
URL: <http://www.gnu.org/software/gsl/>
Required for simulation-quality random number generation and spline interpolation.

```
| tar zxvf gsl-1.16.tar.gz
| cd gsl-1.16
| mkdir build
| cd build
| ../configure CC="gcc" CXX="g++"
| make
| [sudo] make install
```

- **Check** — a unit testing framework for C
URL: <http://check.sourceforge.net/>
Optional, but required to build and run tests.

```
| tar zxvf check-0.10.0.tar.gz
| cd check-0.10.0
| mkdir build
| cd build
| ../configure CC="gcc" CXX="g++"
| make
| [sudo] make install
```

Your system must have these libraries installed prior to compilation of SESE.

To utilize SESE's capability for parallel computation, your system will also need to provide the following:

- **Message Passing Interface (MPI)** implementation, such as OpenMPI
(URL: <http://www.open-mpi.org/>) or MPICH2 (URL: <http://www.mpich.org/>)
- C/C++ compilers with support for **OpenMP** (such as gcc/g++; URL: <http://gcc.gnu.org/>)

Using MPI allows SESE to divide its work load across multiple processes on a single multi-processor machine or on multiple machines connected by a network. OpenMP allows for finer-grained, shared-memory parallelism to utilize multiple cores within a single machine.

SESE has been extensively tested using the GNU GCC compilers gcc/g++, both with and without OpenMP support. As such, it is recommended to use the GCC compilers. Alternate compilers, such as clang, part of the LLVM (llvm.org) infrastructure, should be compatible, but are not completely tested.

Additionally, you may find one or more of the following open-source tools useful for data visualization and processing:

- **ParaView** — an open-source data analysis and visualization application, geared toward analysis of very large datasets

- **gnuplot** — a portable command-line driven graphing utility
- **Python** with NumPy/SciPy — useful for writing scripts to generate and manipulate input/output data
- **GNU Data Language (GDL)** — an open-source interactive data analysis and visualization application, compatible with IDL
- **Interactive Data Language (IDL)** — a commercial interactive data analysis, visualization, and numerical processing language sold by Exelis (Harris).
- **MATLAB** — a commercial interactive data analysis, visualization, and processing language sold by MathWorks.

A.2 Compilation and Installation

Assuming that you are starting from a fresh distribution of version 2.2.0 of SESE, this section will detail the steps necessary to compile and install the program and its utilities. After unzipping the distribution, we recommend building SESE in a subdirectory of the main directory:

```
tar zxvf sese-2.2.0.tar.gz
cd sese-2.2.0
mkdir mybuild
cd mybuild
```

SESE uses the GNU build system and is thus configured and compiled with the following sequence of commands:

```
./configure [options]
make
[sudo] make install
```

It is likely that you will need/want to provide some options on the configuration line, depending on your system and installation preferences. For example, the default installation location will usually be a system-wide location (typically `/usr/local`), for which user write privileges may not be available. For system-wide installation, the `sudo` command can precede the `make install` command. Otherwise, to install SESE to a location in your home directory, you may change the installation prefix:

```
./configure --prefix="$HOME/sese"
```

For more details about the possible configuration options:

```
./configure --help
```

In addition to the standard command line options, the configuration script provides the following options:

Table A-1. Configuration script options

Option	Possible values	Comment
<code>-with-mpi</code>	auto (default)	If an MPI implementation is found on your system, compile SESE with support process-level parallelization via MPI.
	yes	Require compilation with MPI. If MPI is not found, configure will exit with an error.
	no	Disable MPI parallelization.
<code>-with-openmp</code>	auto (default)	If OpenMP is found on your system, compile SESE with support for thread-level parallelization via OpenMP. Number of threads can be determined at run-time via the <code>-t</code> command line option.
	yes	Require compilation with OpenMP. If OpenMP is not found, configure will exit with an error.
	no	Disable OpenMP parallelization.
<code>-enable-extended-tests</code>	yes	When running <code>make check</code> , perform integration tests to verify accuracy of simulation results in some simple test cases. These tests may take a significant amount of time to run.
	no (default)	<code>make check</code> only performs simple module tests.

If some required libraries are installed in a non-standard location, or if you have multiple versions of a library installed, you may need to explicitly tell the configure script where to find them. You can do this by passing the following additional environment variables on the `configure` line:

```
|| ./configure CPPFLAGS="-I/path/to/include" LDFLAGS="-L/path/to/lib"
```

By default, the `libyaml-cpp` library is built only as a static library.¹ If you want to link against a particular static library, try passing it directly via the `LIBS` variable:

```
|| ./configure LIBS="/path/to/lib/libmystatic.a"
```

The core SESE library is written in C/C++ and must be compiled with C/C++ compilers. You can explicitly specify which C/C++ compilers to use via the `CC` and `CXX` variables. Remember that the

¹To compile `libyaml-cpp` as a shared library, compile with: `cmake -DBUILD_SHARED_LIBS=ON`

same compilers must be used for the external libraries:

```
|| ./configure CC="gcc-4.6" CXX="g++-4.6"
```

For example, if you wish to compile with support for MPI parallelization, you may want to pass your MPI compiler explicitly:²

```
|| ./configure --with-mpi CC="mpicc" CXX="mpic++"
```

By default, SESE is compiled with the optimization flag `-O3`. You can change this flag or pass your own optimization flags via the `CFLAGS` and `CXXFLAGS` variables. For example, to disable optimization:

```
|| ./configure CFLAGS="-O0" CXXFLAGS="-O0"
```

If all goes well in the configuration step, you should be able to compile and install SESE as usual:

```
|| make  
|| [sudo] make install
```

If writing to your installation directory requires root privileges (e.g. for a system-wide installation), you may need to install with `sudo make install` instead.

A.3 Unit Testing and Extended Integration Tests

To verify that SESE works correctly on your system, a range of unit tests has been included to ensure functionality of critical code modules. These unit tests can be run with `make check`. Additionally, if you pass the flag `-enable-extended-tests` to the `configure` script, SESE will perform extended integration tests to verify simulation results against analytic predictions for some simple test cases. If `configure` has detected that you have the Check unit testing framework installed, you can build and run these tests with

```
|| make check
```

A.4 Installation On High-Performance Computers

The installation of SESE in a parallel (cluster) HPC environment follows the same basic steps but requires some additional organizational steps to ensure proper distribution of resources across all computational nodes. A SESE user³ compiled the following notes for various HPC environments. A couple of environment variables are set prior to the installation:

- `DEPENDENCY_DIR` - set to the directory where libraries and headers are found
- `INSTALL_DIR` - set to the directory where the libraries and program are to be installed

In addition, items in brackets will depend on your particular configuration and user setup.

²If you encounter problems during compilation involving the OpenMPI C++ bindings, try adding the flag `-DOMPI_SKIP_MPICXX` to `CPPFLAGS`.

³Thanks to Dr. Jeremiah Babcock for sharing his notes and experience.

A.5 Cluster Computer

To compile and install SESE on a cluster computer, first the user must log in, preferably to the master (head) node of the cluster:

```
|| ssh <user_name>@<ip_address>
```

The environment variable `DEPENDENCY_DIR` should be set to a location in shared storage. After the directory structures have been established, SESE and its dependency tarballs should be transferred via SCP to the corresponding `DEPENDENCY_DIR`. This can be done by:

```
|| scp <tarball_name>.tar.gz <user_name>@<ip_address>:\$DEPENDENCY_DIR
```

To build SESE with MPI support, you will need to have `mpich2.1.4` or higher loaded/enabled using the command "module". Run this command to make the MPI libraries before building SESE.

```
|| module load mpich/2.1.4
|| ./configure --prefix=$INSTALL_DIR --with-openssl
||     --enable-extended-tests --with-mpi CC="mpicc"
||     CXX="mpic++" CFLAGS="-std=c99" CPPFLAGS="-I/$DEPENDENCY_DIR/include"
||     LDFLAGS="-L/$DEPENDENCY_DIR/lib" 2>&1 | tee configure.out
|| make 2>&1 | tee make.out
|| make check
|| [sudo] make install
```

To compile using OpenMPI support for parallel threads on each node, you will need to load the latest openMPI module on the cluster first. If an error is thrown about MPI C++ bindings, you will need to disable the OpenMPI C++ bindings by adding the flag `DOMPI_SKIP_MPICXX` to your `CPPFLAGS` environmental variable.

```
|| module load openmpi/<latest_version>
|| ./configure
||     --prefix="$INSTALL_DIR" --with-openssl=yes
||     --enable-extended-tests=yes --with-mpi CC="mpicc" CXX="mpic++"
||     CFLAGS="-std=c99" CPPFLAGS="-I/$DEPENDENCY_DIR/include"
||     LDFLAGS="-L/$DEPENDENCY_DIR/lib" 2>&1 | tee configure.out
|| make CPLUS_INCLUDE_PATH="$DEPENDENCY_DIR/lib"
||     LIBRARY_PATH="$DEPENDENCY_DIR/lib" 2>&1 | tee make.out
|| make check
|| [sudo] make install
```

A.6 DoD HPC - Garnet (CRAY XE6 Cluster)

To compile SESE's dependencies `gsl`, `yaml-cpp`, and `check`, the `CC` and `CXX` flags must be passed, along with the correct loading of modules.

- check and gsl setup

```
|| module swap PrgEnv-pgi PrgEnv-gnu
|| module load gcc
|| export CC=gcc CFLAGS="-fPIC -O3" LDFLAGS="-O3" CXX=g++ CXXFLAGS="-O3"
|| ./configure --prefix="$DEPENDENCY_DIR" --enable-static
```

```

2>&1 | tee configure.out
make 2>&1 | tee make.out
make -k check 2>&1 | tee make_check.out
[sudo] make install

```

- **yaml-cpp setup**

```

module swap PrgEnv-pgi PrgEnv-gnu
module load gcc cmake/2.8.12.2-gnu
export CC=gcc CFLAGS="-fPIC -O3" LDFLAGS="-O3" CXX=CC CXXFLAGS="-O3"
mkdir build
cd build
cmake -DBoost_INCLUDE_DIR=$DEPENDENCY_DIR/include
      -DCMAKE_INSTALL_PREFIX:PATH=$DEPENDENCY_DIR ..
make 2>&1 | tee make.out
[sudo] make install

```

- **SESE**

```

module swap PrgEnv-pgi PrgEnv-gnu
module load gcc
./configure --prefix="$INSTALL_DIR" --with-openmp=yes --with-mpi=yes CC="
cc"
CXX="CC" CFLAGS="-dynamic" CXXFLAGS="-dynamic"
CPPFLAGS="-I$DEPENDENCY_DIR/include"
LDFLAGS="-L$DEPENDENCY_DIR/lib" 2>&1 | tee configure.out
make 2>&1 | tee makesese.out
make -k check 2>&1 | tee make_check.out
[sudo] make install

```

A.7 DoD HPC - Excalibur (CRAY XC40 Cluster)

The sequence of commands for building SESE on Excalibur are similar to those used on the Garnet HPC:

- **SESE**

```

module swap PrgEnv-intel PrgEnv-gnu
module load gcc
./configure --prefix="$INSTALL_DIR" --with-openmp=yes --with-mpi=yes CC="
cc"
CXX="CC" CFLAGS="-dynamic" CXXFLAGS="-dynamic"
CPPFLAGS="-I$DEPENDENCY_DIR/include"
LDFLAGS="-L$DEPENDENCY_DIR/lib" 2>&1 | tee configure.out
make 2>&1 | tee makesese.out
make -k check 2>&1 | tee make_check.out
[sudo] make install

```

A.8 DoD HPC - Thunder (SGI ICE X Cluster)

SESE can be compiled on the Thunder HPC using `gcc`, but better optimization and performance is likely when using the Intel compilers `icc` and `icpc`. These compilers are already loaded, as well as the MPI protocols `mpt/2.12`. Some flags must be added to compile successfully.

- SESE

```
./configure --prefix="$INSTALL_DIR" --with-openmp=yes --with-mpi=yes  
CC="icc" CXX="icpc" CFLAGS="-lmpi -fast -O3"  
CXXFLAGS="-lmpi -fast -O3" CPPFLAGS="-I$DEPENDENCY_DIR/include"  
LDFLAGS="-L$DEPENDENCY_DIR/lib" 2>&1 | tee configure.out  
make 2>&1 | tee makesese.out  
make -k check 2>&1 | tee make_check.out  
[sudo] make install
```

APPENDIX B - CONFIGURATION AND USE

B.1 Overview

To define the simulated physical scenario completely, SESE needs several pieces of data that answer the following questions:

1. What materials are simulated and how do they behave at different temperatures and wavelengths?
2. What is the spatial arrangement of those materials?
3. What is the shape of incident radiation sources?
4. What is each source's wavelength, power, position, orientation, etc.?
5. Do these parameters change over time? What temporal sequence of events should occur during the simulation?
6. What output data are desired?
7. What simulation parameters (desired error tolerances, maximum time step, etc.) should SESE use?

The following input files answer these questions, respectively (see Figure B-1):

1. Material library and material definitions (YAML text files)
2. Object and volume fraction files (a three-dimensional model in binary format)
3. Source file(s) (one or more two-dimensional planar distributions in binary format)
- 4–7. Configuration file (a YAML text file)

To reduce clutter, separation of each type of file is encouraged by using a directory structure like the following:

```
|| config/   material/   object/   output/   source/
```

The contents of each type of file is explained in more detail below. In addition, look at the folders in the "examples/" directory of the distribution for recommended organization strategies.

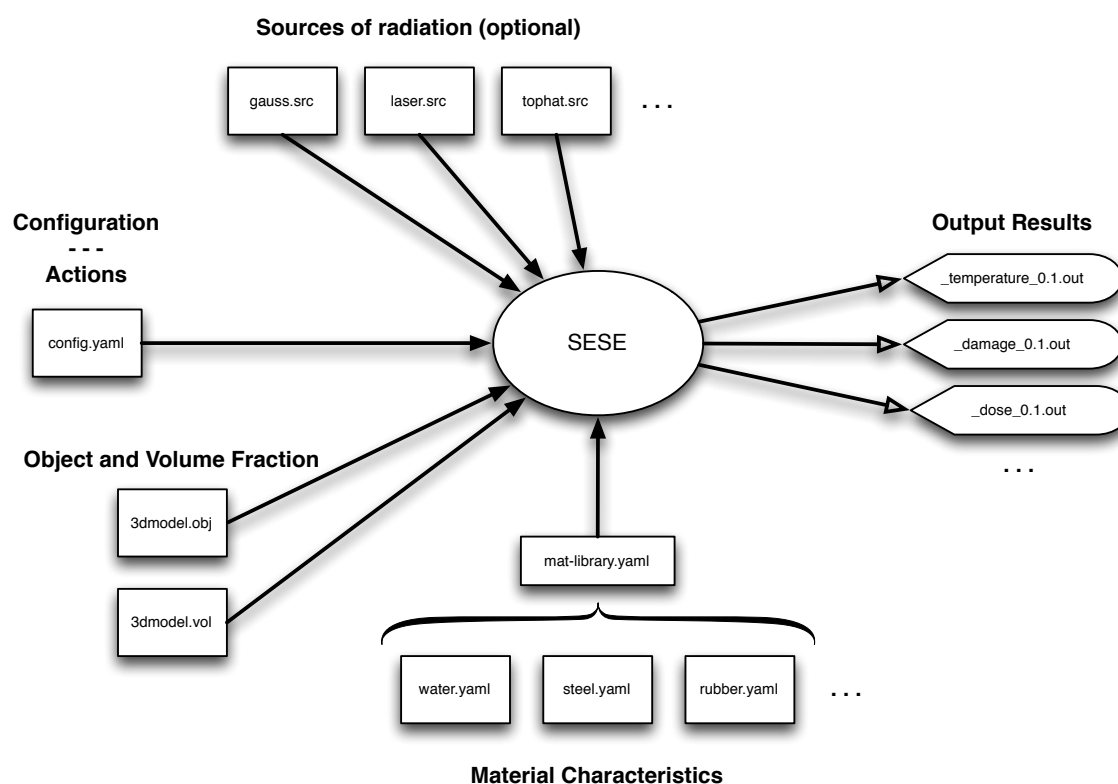


Figure B-1. Block diagram showing input and output files for a typical SESE simulation run

B.2 A Word On The YAML Format

SESE's material files and configuration files are written in YAML, a simple, human-readable format for data serialization.⁴ YAML avoids the use of markup characters that clutter the data and reduce readability; instead, YAML encodes structure using whitespace and indentation. However, this means that a user must be careful to use proper indentation, otherwise incorrect or unexpected behavior may result. SESE provides the `seseconfigcheck` utility as an aid to the user in making sure their YAML configuration files are valid prior to running SESE.

Whitespace should consist of space characters only, not tabs. It is recommended to use two space characters per indentation level, which is the convention used in the examples included in the SESE distribution.

It is useful to become familiar with YAML's main constructs to understand how a YAML document represents data. There are four primary YAML constructs, described below. Comments are possible anywhere in a YAML document using the hash '#' character – text from the hash to the end of the current line is ignored.

1. **Scalars:** simple fragments of text; e.g. `3.5` or `"filename.txt"`
2. **Maps:** unordered sets of key-value pairs, where the key is a scalar and the value may be any valid YAML construct. Maps may optionally be contained within curly braces, with key and value associated by a colon; e.g. `{ name: "John" , age: 25 }` or simply

```
name: "John"
age : 25
```

3. **Sequences:** ordered lists of YAML constructs, either contained within square brackets or enumerated with dashes; e.g. `[ one, two, three ]` or

```
- scalar_one
- "scalar2.0"
- {map_three: 3.0E-15}
```

4. **Documents:** a YAML construct delimited by three dashes in a row (---). For example, the following listing has four YAML documents. The first document is a single map; the second is a sequence of maps containing information about the platonic solids, where the value of the 'info' key in each map is another map; the third is a sequence of sequences of three scalars; and the fourth document contains the lone scalar 'Bob'.

```
# document 1
{type: vegetable, color: green}    #single map
---
#document 2
- {name: tetrahedron, info: {edges: 6, vertices: 4, faces: 4}}
- {name: cube, info: {edges: 12, vertices: 8, faces: 6}}
- {name: octahedron, info: {edges: 12, vertices: 6, faces: 8}}
- {name: dodecahedron, info: {edges: 30, vertices: 20, faces: 12}}
```

⁴For more information on the YAML 1.2 specification, see: <http://www.yaml.org/spec/1.2/spec.html>

```

- {name: icosahedron, info: {edges: 30, vertices: 12, faces: 20}}
---
#document 3
- [red, blue, white]
- [1, 2, 3]
---
#document 4
Bob

```

B.3 Material Library and Definitions

SESE allows the user to define the optical, thermal and chemical properties of various materials in the simulation. Entries in a *material definition file* define the dynamic physical properties for a given material. A central *material library* refers to each material definition.

B.4 Material Library

Each material is associated with a unique, positive 16-bit integer called its ‘material ID’. A material library keeps the list of all material definitions and their associated IDs, which is a YAML file containing a sequence of maps such as the following:

```

- id: 1
  file: water.yaml
- id: 2
  file: glass.yaml
- id: 5
  file: plastic.yaml
# ...

```

Material ID 0 is reserved to denote the part of the computational domain external to the object and may not appear as an entry in the material library. The region external to the object is transparent, non-thermally-conductive, and non-chemically-reactive. At boundaries between regions of material ID 0 and regions of non-zero material ID, various thermal boundary conditions may be imposed.

B.5 Material Definition File

The physical properties of a material are allowed to vary as a function of wavelength λ , temperature T , and the fractional concentration c of ‘native’ material remaining that has not undergone a chemical change due to thermal damage (which is related to a function of the temperature at all previous times, i.e. the temperature ‘history’). The corresponding material definition file specifies these properties for each material, which is a YAML file consisting of several sections for optical, thermal, and chemical properties.

All quantities in the material file are specified in SI-MKS units, with temperatures in degrees Celsius. The only exception to this rule is the optical property wavelength, which must be specified in nanometers.

Optical properties may be dependent on wavelength, temperature and damage, whereas thermal properties are dependent only on temperature and damage, while chemical properties are dependent on temperature alone. In the current version of SESE, the available material properties, their units, and their dependence on the independent variables are listed in the Reference section.

Any of the material properties may be omitted from the material file, in which case they will be assigned reasonable, constant default values; for example, zero absorption, zero scattering, zero thermal diffusion, etc. The following example material file exhibits the default material parameters:

```

1 | optical:
   |   - wavelength: 0          # nm
   |     refractive_index: 1
   |     absorption: 0         # 1/m
5  |     scattering: 0          # 1/m
   |     scattering_g: 0
   |
   | thermal:
   |   conductivity: 0         # W/m*K
10 |   density: 1              # kg/m^3
   |   specific_heat: 1        # J/kg*K
   |   metabolic_rate: 0       # J/m^3
   |   perfusion: 0            # 1/s
   |   heat_transfer_coeff: 0  # W/m^2*K
15 |   emissivity: 0
   |   evap_coeff: 0           # W/m^2*Pa
   |
   | chemical:
   |   activation_energy: 1e20 # J/mol
20 |   arrhenius_prefactor: 0   # 1/s

```

The value of each material property at one or more temperatures may be specified explicitly in the material definition file in a tabular fashion, as in the following excerpt:

```

1 | # ...
   | thermal:
   |   conductivity: 0.58
   |   density: |
5  |     spline
   |       0 999.8
   |       4 1000
   |      10 999.7
   |      20 998.2
10 |      90 965.3
   |      100 958.4
   |   specific_heat: |
   |     polynomial
   |       4200.0 -0.02 0.0004
15 |   evap_coef: |
   |     linear
   |       0 0.5
   |      50 0.6
   |     100 0.8
20 | # ...

```

This example defines the temperature dependence of the density over the range 0–100°C, the specific heat as a quadratic polynomial in temperature, and the evaporation coefficient as a piecewise linear function of temperature, while the thermal conductivity is constant for all temperatures. For temperatures between the specified data points, SESE will perform interpolation to determine the value of the material property. By including the `spline` keyword in the first line of the table, cubic spline interpolation will be used for values between the given data points.

The following keywords are available to specify the interpolation type:

- `linear` — linear piecewise interpolation
- `polynomial` — polynomial evaluation of order (number of coefficients -1)
- `spline` — cubic spline interpolation

By default, SESE uses linear interpolation. For the spline and linear interpolation types, outside the specified range, the material property will simply take on the value of the nearest endpoint. The data points must be ordered by ascending temperature, otherwise the behavior of the interpolation is undefined.

Note the use of the vertical bar (“|”) character to split the tabular input over several lines. Any text following the vertical bar symbol will be interpreted literally, so you should not include any comments or extraneous text within these lines. Also carefully note the change in indent level, which signals the end of the tabular input.

SESE also supports simple simulation of a ‘damage’ chemical reaction in which the properties of a material change irreversibly due to exposure to elevated temperatures over a period of time. When the temperature dependence of the material in its native state (containing only reactants) and in its damaged state (containing only products) are specified, the material properties can be interpolated at any level of damage and temperature according to the relative concentrations of native and damaged material computed using the Arrhenius damage model. The native and damaged properties can be specified independently for each property via the map keys “0” and “1”, respectively. Both of these map keys must be present; it is not sufficient to include just one. In future versions, SESE may support multiple damage states beyond these two.

For example, the following defines a material that, in its native state, becomes more transparent as temperature increases, but has a large, constant absorption in its damaged state:

```
optical:
- wavelength: 2000
  absorption:
    0: |
      0    270
      40   230
      60   190
      100  140
    1: 5000
```

The Arrhenius damage model controls the transition between the native and damaged state, whose parameters appear in the `chemical` section of the material file. A very large activation energy (10^{20}) and a zero Arrhenius-prefactor force a material to remain in the native state.

Due to the complicated dependence of real material properties on incident radiation wavelength, interpolation between wavelengths is not performed. Radiation sources used in the simulation must be specified to have a wavelength (rounded to an integer number of nm) that exactly matches the wavelength data specified in the material definition files, otherwise an error will result.

B.6 Object and Volume Fraction Files

If the object to be modeled can be constructed by combining geometric primitives (half-spaces, boxes, cylinders, and spheres) using boolean set operations (union, intersection, and complement), then it may be generated using the `makegeom` library included with SESE. If your object is more complicated, you may prepare it by hand or computationally using a tool such as Python, MATLAB or IDL/GDL to construct a three-dimensional representation of the object, write out the corresponding sequence of material IDs as text, and then use the `raw3dfromtxt` tool to convert it to the SESE RAW3D format.

As of SESE version 2.1.0, the `makegeom` library supports generation of both object and volume fraction files.

B.7 Using `makegeom`

The `makegeom` C++ library included with SESE simplifies generation of objects formed by geometric primitives, such as spheres, ellipsoids, cylinders, boxes, etc. Use of the `makegeom` library requires writing and compiling your own C++ program that call the various `makegeom` functions and write the object material IDs and volume fractions to SESE-compatible files.

The following simple example generates an object which is the intersection of a sphere and a cube, contained within a $(2\text{ cm})^3$ volume (see Figure B-2):

```

1  /* demo.cpp */
   #include "makegeom.h"

5  int main()
   {
       Sphere s(0,0,0, 4e-3);           // centered at 0,0,0, radius 4 mm
       Box b(-3e-3,-3e-3,-3e-3, 3e-3,3e-3,3e-3); // centered at 0,0,0, 6 mm on side
       Region s_intersect_b = s.Intersect(b);

10  Object o;                           //instantiate an object
       o.set_bbox(-1e-2,-1e-2,-1e-2, 1e-2,1e-2,1e-2); // opposite corners of (2 cm)
           ^3 bbox
       o.set_dims(200, 200, 200); // discretization: 200x200x200
       o.push(s_intersect_b, 1);      // material ID 1
15  o.write_obj("demo.obj");          // write object file
       o.write_vol("demo.vol");       // write volume fraction file

       return 0;

```

```
|| }
```

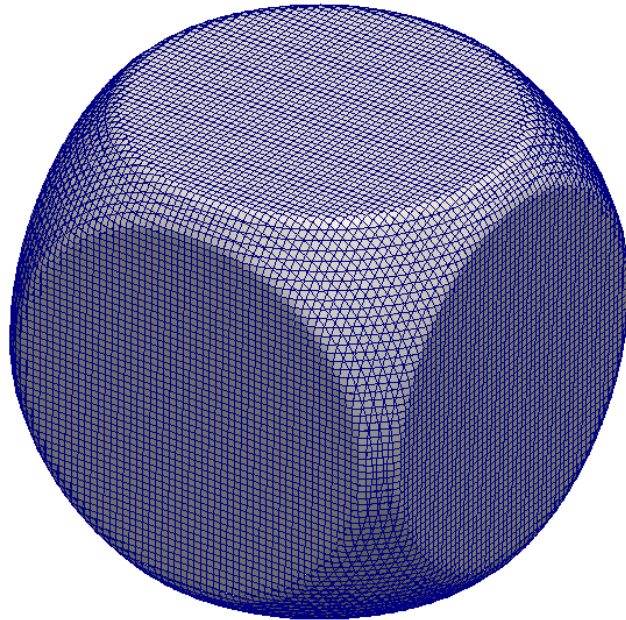


Figure B-2. Intersection of a cube and a sphere, generated by the above example. PLIC plane representation generated by the `volpoly` utility and visualized in ParaView

Each primitive (e.g. Sphere, Box, Halfspace) is defined by a number of parameters that specify its dimensions and location in space.

```
|| Sphere a(x,y,z, r);           // center and radius
|| Box b(x1,y1,z1, x2,y2,z2);    // coordinates of corners
|| Halfspace c(x,y,z, nx,ny,nz); // any point on boundary and outward normal
||                               vector
```

A new Region can be defined by combining two primitives or regions a and b by using the operations Union, Intersect, and Minus:

```
|| Region r1 = a.Union(b);
|| Region r2 = a.Intersect(b);
|| Region r3 = a.Minus(b);
```

The complement of a region is given by Not (which takes no argument):

```
|| Region p = c.Not();
```

Any number of primitives or regions can be combined in this manner:

```
|| Region q = a.Intersect(b).Minus(c.Union(d).Union(e));
```

Once the primitives/regions composing the object have been defined, the material ID occupying each region is defined as each regions is “pushed” onto the object. If there is some location in space

that is contained by more than one pushed region, it will contain the material ID of the last region pushed; hence, the suggestive term “push”. The corners of the bounding box containing the object is specified by calling `set_bbox()` and the discretization is specified by calling `set_dims()`. Finally, the material ID and volume fraction data are written to file.

```
Object o;
o.push(p, 2); // material ID 2
o.push(q, 7); // material ID 7
o.push(r, 3); // material ID 3
o.set_bbox(x1,y1,z1, x2,y2,z2);
o.set_dims(nx, ny, nz);
o.write_obj("demo.obj");
o.write_vol("demo.vol");
```

During the installation step, SESE will have installed the `makegeom.h` header in your installation directory. To compile and link your program with the `makegeom` library, you can use:

```
g++ -I$HOME/sese/include -o demo demo.cpp
```

Make sure to include linker flags for any other libraries you may have used in your program (e.g. `-lm` if you have used the math library). Now, when the program executes, it generates the object and volume fraction files:

```
./demo
```

The dimensions and spacings of the generated files can be easily inspected by using the `raw3dinfo` tool, and the object itself can be visually inspected in ParaView, for example, by using the `raw3dtoxdmf` tool.

B.8 Using IDL/GDL

Objects formed from geometric primitives can be similarly ‘drawn’ by an IDL/GDL script using the set of functions included in the `share/sese/examples/obj-idl-makeobject` directory. This allows the user to create object files from simple scripts of geometric primitives without having to write and compile a program. A script to generate a simple ‘tin man’ model is provided as an example.

B.9 Using The RAW3D Tools

Object files can also be created by writing a custom script that simply prints each voxel’s material ID in the appropriate order and piping its output to the `raw3dfromtxt` tool provided with SESE. Numeric values must be printed in *x*-increasing order and separated by whitespace.

The `raw3dfromtxt` tool has the ability to detect the dimensions of the data from its formatting automatically: a newline signifies the end of a row of data (and hence an increment in the *y*-coordinate), and a blank line signifies the end of a planar slice of data (and hence an increment in the *z*-coordinate).

For example, the following numeric data:

```
# in.txt
```

```

1 1 1 1
1 1 1 1
1 1 1 1

2 2 2 2
2 2 2 2
2 2 2 2

```

corresponds to an object with dimensions $4 \times 3 \times 2$ with materials 1 and 2 in the bottom and top z-layers, respectively, corresponding to material IDs 1 and 2 specified in a material library file. These formatted numeric data pass to the tool `raw3dfromtxt` to produce a SESE object file:

```

raw3dfromtxt -t int16 -h 1e-3x1e-3x1e-3 out.obj < in.txt

```

This will produce an object file named `out.obj` containing the input values from `in.txt` stored as 16-bit integer values (specified by the mandatory `-t` option) with a uniform grid spacing of 10^{-3} meters in each axis (specified by the `-h` option). Alternatively, the `-l` option used in place of the `-h` option specifies the physical size of the entire object (in meters):

```

raw3dfromtxt -t int16 -l 4e-3x3e-3x2e-3 out.obj < in.txt

```

The dimensions $4 \times 3 \times 2$ are automatically detected from the formatting of the values in `in.txt`. If the input is not formatted in this manner but is whitespace-delimited, the `raw3dfromtxt` tool can still be used by explicitly specifying the dimensions with the `-n` option:

```

raw3dfromtxt -t int16 -h 1e-3x1e-3x1e-3 -n 4x3x2 out.obj < in.txt

```

Of course, the number of values in the input dataset must match those expected from the given dimensions.

The dimensions, spacings, size and data type of the generated object can be verified with the `raw3dinfo` tool:

```

$ raw3dinfo out.obj
type (width) nx ny nz hx [m] hy [m] hz [m] lx [m] ly [m] lz [m] file
-----
int16      (2)  4  3  2  1e-03  1e-03  1e-03  4e-03  3e-03  2e-03 out.obj

```

APPENDIX C - RUNNING SESE SIMULATIONS

C.1 Creating, Running, and Analyzing a SESE Simulation

In this section, we walk through the setup and creation of a SESE simulation for a hypothetical scenario with two laser sources and a material damaged by heat. This simulation is in the `examples` folder of the SESE distribution, in the folder `sim-thermochromic`. It may be more instructive for you to follow along, create the files, and run the simulation in a separate directory.

In this simulation, we will have a spherical object with a diameter of 20 cm, with a 1 cm thick planar layer of different material passing through the center. The outer part of the sphere is composed of a variant of polycarbonate, and the inner layer is a thermochromic glass material. Start by setting up a directory to hold the input files and output data for the simulation.

```
cd <directory where you want the simulation folder to be>
mkdir sim-thermochromic
cd sim-thermochromic
mkdir config
mkdir material
mkdir object
mkdir output
```

C.2 Creation Of The Object File

We will use the `makegeom` library to create the object and `volfrac` files that SESE will need for this simulation. It is usually helpful to sketch the geometry to make sure that dimensions and placement are correct. Something simple, like Figure C-1, is sufficient.

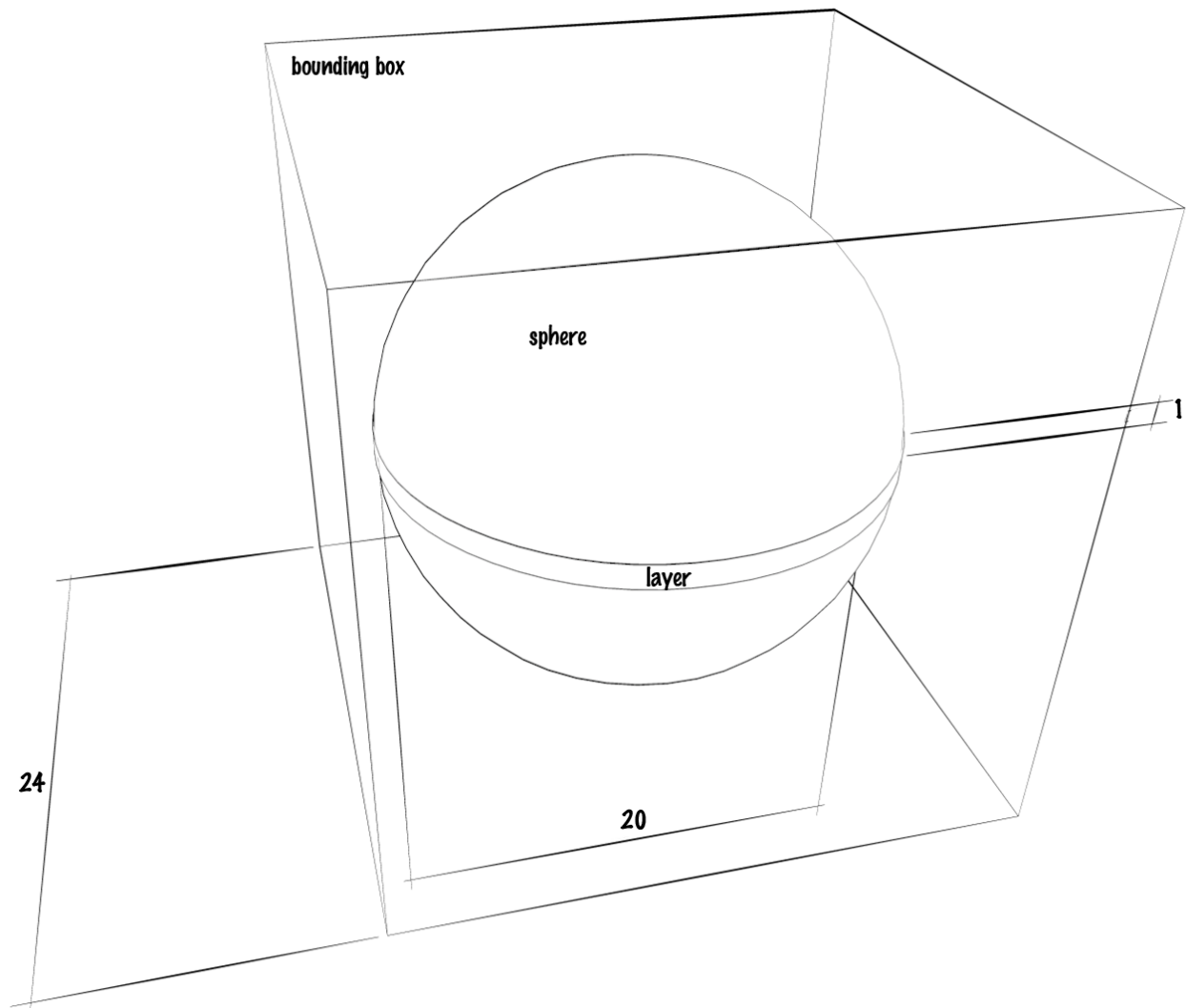


Figure C-1. Sketch of spherical object with planar layer, in bounding box domain

The geometry of this scenario is simple, and the C++ object creation program is straightforward. Decisions were made to put the center layer in the $x - y$ plane, make it material ID 2, and to make the sphere material ID 1. Remember that SESE expects everything in MKS units, so dimensions are in meters. Another thing to remember is that the origin of the Cartesian coordinate system may be in a different location in the object creation program and in SESE. In SESE, the object is always loaded into the first octant of the 3D coordinate system (such that all voxels in the domain are located at positive values of x, y, z). In the `makegeom` environment, the location of the origin is not important, and the object can be constructed in any location that is convenient. The C++ source file that creates the object and volfrac file is `sphere-layer.cpp`, and it should be saved in the `object` subdirectory.

```
1 //
  //  sim-thermochromic
  //  object creation program
  //
```

```

5  #include <makegeom.h>

int main(void)
{
    // dimensions of object
10  double  bbs = 0.24;                //side of bounding box
    double  spherediam = 0.2;          //20 cm diameter sphere
    double  thk = 0.01;                //central layer thickness

    // dimensions of simulation discretization
15  unsigned int  nxyz = 100;          //cubic voxels

    // boolean geometry construction
    Sphere  sph(0, 0, 0, spherediam / 2.0);    //sphere centered at origin
    Box      b(-bbs, -bbs, -thk/2.0, bbs, bbs, thk/2.0);
20  Region  layer = sph.Intersect(b);          //central layer is part of box
        in sphere

    // instantiate the mesh
    Object  mesh;                          //create the mesh object and
        set bounding box
25  mesh.set_bbox(-bbs/2.0, -bbs/2.0, -bbs/2.0, bbs/2.0, bbs/2.0, bbs/2.0);
    mesh.set_dims(nxyz, nxyz, nxyz);          //set the discretization

    // put materials in the mesh
    mesh.push(sph, 1);                      //insert sphere of material 1
    mesh.push(layer, 2);                      //insert the layer of material
        2
30

    // output the object and volfrac files
    mesh.write_obj("sphere-layer.obj");        //write object file
    mesh.write_vol("sphere-layer.vol");        //write volfrac file

35  return 0;                            //successful exit
}

```

From the object subdirectory, compile the program `sphere-layer.cpp` and execute it to create the object file and the volfrac file.

```

| cd object
| g++ sphere-layer.cpp -o sphere-layer
| ./sphere-layer

```

C.3 Creation Of The Material Files

Now that the object has been created, we need to create a material library file that contains the association between the material ID number and the corresponding material file. The library file is a simple YAML sequence of maps with the keywords `id:` and `file:`. The library file and the two material files are placed in the `materials` subdirectory.

```

| - id: 1
|   file: polycarbonateOD2015.yaml
| - id: 2

```

```
|| file: thermochromicglass.yaml
```

The material file `polycarbonateOD2015.yaml` for the Makrolon OD2015 polycarbonate polymer appears below. We have included optical characteristics for the two wavelengths we are going to use in the simulation, $1.37\ \mu\text{m}$ and $0.5\ \mu\text{m}$. Note that the polymer has a moderate absorption at the infrared wavelength, but is only weakly absorbing at the green wavelength. We do not anticipate damaging the polycarbonate, so the chemical parameters are set to their default values. (These parameters could have simply been left out of the material file and would have been set to these same values.)

```
1 | # material file for clear polycarbonate polymer
  | # Makrolon OD2015
  | #
  | optical:
5 |   - wavelength: 500                # nm
    |     refractive_index: 1.585
    |     absorption: 0.5              # 1/m
    |     scattering: 0.1              # 1/m
    |     scattering_g: 0.5
10 |   - wavelength: 1370              # nm
    |     refractive_index: 1.555
    |     absorption: 5.3              # 1/m
    |     scattering: 0.1              # 1/m
    |     scattering_g: 0.5
15 | thermal:
    |   density: 1210.0                # kg/m^3
    |   specific_heat: 1250.0          # J/kg/K
    |   conductivity: 0.21             # W/m/K
    |   perfusion: 0.0                # sec^-1
20 |   metabolic_rate: 0.0              # W/m^3
    |   heat_transfer_coeff: 0.0       # W/m^2/K
    |   evap_coeff: 0.0                # W/m^2/Pa
    |   emissivity: 0.3
25 | chemical:
    |   arrhenius_prefactor: 0.0       # 1/sec
    |   activation_energy: 1.0E20      # J/mol
```

The thermochromic glass material is a hypothetical doped glass material that has a very high absorption coefficient in the infrared. The glass is doped with visible-band (VIS) chromophores that change their absorption coefficient as a function of temperature. The glass absorbs VIS light weakly at room temperature, but the absorption coefficient increases sharply as temperatures rise up to $40\ ^\circ\text{C}$. As long as the temperature stays below about $40\ ^\circ\text{C}$, the absorption is reversible with temperature, *i.e.* the absorption will decrease as temperature decreases. At higher temperatures, however, the chromophores are damaged and the material irreversibly becomes non-absorbing. The absorption in the infrared band remains large, however, independent of temperature or damage to the chromophores. These characteristics of the thermochromic glass can be modeled with the following material file, `thermochromicglass.yaml`.

```
1 | # material file for thermochromic glass
```

```

#
optical:
- wavelength: 500          # nm
  refractive_index: 1.522
  absorption:
    0: |
      linear
      10.0 1.0
      20.0 2.0
      30.0 10.0
      40.0 50.0
    1: 0.1          # damaged
  scattering: 0.01      # 1/m
  scattering_g: 0.5
- wavelength: 1370        # nm
  refractive_index: 1.504
  absorption: 300.0      # 1/m
  scattering: 0.0        # 1/m
  scattering_g: 0.5

thermal:
  density: 2510.0        # kg/m^3
  specific_heat: 858.0   # J/kg/K
  conductivity: 1.11     # W/m/K
  perfusion: 0.0         # sec^-1
  metabolic_rate: 0.0     # W/m^3
  heat_transfer_coeff: 0.0 # W/m^2/K
  evap_coeff: 0.0        # W/m^2/Pa
  emissivity: 0.2

chemical:
  arrhenius_prefactor: 1.48E70 # 1/sec
  activation_energy: 4.34E5     # J/mol

```

Note that the VIS band ($0.5\ \mu\text{m}$) absorption coefficient has a tabular piecewise linear temperature dependence for the "0" (native undamaged) state, and a constant absorption coefficient in the "1" (damaged) state. Arrhenius kinetics specifies damage (see Section 3.2 for more information), that is, the damage reaction rate k as a function of temperature T is assumed to follow the relation

$$k = A \exp\left(-\frac{E_a}{RT}\right) \quad (99)$$

In many cases, data for decomposition or other reaction rates are not published. In those cases, the values of the Arrhenius parameters A and E_a are needed such that the reaction rate equals 1.0 at a given critical temperature T_r , and the derivative of the rate with respect to temperature, at temperature T_r , is equal to S_r . Recall that a material is damaged if the integrated reaction rate exceeds about 0.5. Thus, for conditions in which the reaction rate is about $1\ \text{sec}^{-1}$, it takes 0.5 sec to damage the material. It can be easily verified that the following relations will provide the Arrhenius constants, given T_r and S_r

$$\ln A = S_r; \quad E_a = R S_r T_r^2 \quad (100)$$

For our thermochromic material, we set the critical temperature to $T_r = 50^\circ\text{C}$, and the slope of the reaction rate to $S_r = 0.5 \text{ sec}^{-1} \text{ K}^{-1}$, resulting in the Arrhenius parameters appearing in the `chemical` section of the material file. A semilog plot of the expected Arrhenius damage rate as a function of temperature appears in Figure C-2.

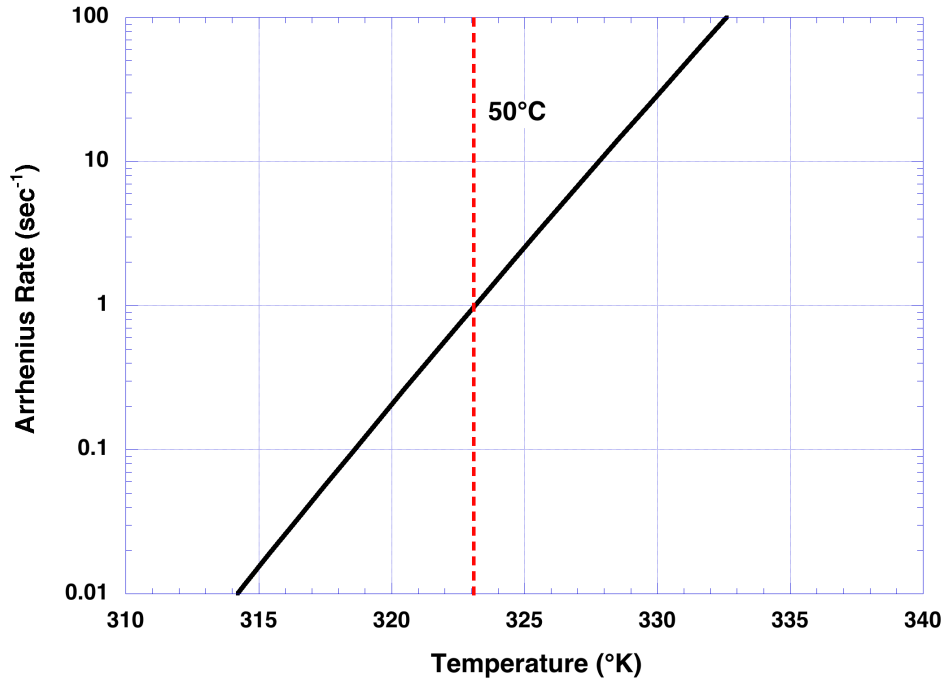


Figure C-2. Thermochromic damage rate as a function of temperature

C.4 Estimating Temporal Response

Now that the object and materials have been defined, we next define the simulation configuration parameters and the action script. From a top-level overview, we want to investigate the transient behavior of the thermochromic glass material under laser irradiation. A good way to do that in the lab is with a pump-probe arrangement, and we will have our simulation mimic that experimental approach. Briefly, a powerful laser (the pump) stimulates the thermal characteristics of the material, and a smaller "probe" laser makes measurements of the material response. In this case, we will choose a pump laser in the SWIR band, because we know that the absorption of the thermochromic glass is large at that wavelength, and does not depend on the temperature or damage. The probe laser is chosen at a VIS wavelength, where the largest response to temperature and damage is expected. The power in the probe laser beam is too small to add any meaningful thermal stimulus to the material. In our simulation, we will pump the thermochromic material with the pump laser, then turn it off and probe the material characteristics with the probe laser. We will do this repeatedly, also plotting the temperature and damage metrics within the simulated object.

First look at the physics of the simulation, and what we are expecting to happen. First, we will have collimated laser beams (pump and probe) that are incident upon a spherical interface (the spherical polycarbonate polymer). We know that the effective focal length of a dielectric interface is

$$f_{eff} = \frac{n_2 R}{n_2 - n_1} \quad (101)$$

where R is the radius of curvature of the dielectric surface, and n_1 and n_2 are the refractive indices of the incident medium (vacuum) and the internal medium (polycarbonate), respectively. We find that the outside surface of the polycarbonate sphere (recall the diameter of the sphere is 0.2 m) has an effective focal length at the pump wavelength of 0.28 m, which means that the pump beam will come to a focus beyond the far side of the sphere. We thus expect the pump beam diameter at the center of the sphere, where the thermochromic material is, to be roughly one-third its initial diameter, or three times its initial irradiance. The absorption of the polycarbonate at the pump wavelength is 5.3 m^{-1} , so we expect attenuation of the pump beam irradiance by a factor of about 0.5 by the time it reaches the thermochromic layer. The attenuation of the pump beam in the thermochromic layer is very high, and for a 1 cm thickness, we expect that about 95% of the incident pump beam will be absorbed in the layer. Thus, beginning with a 5 cm radius beam with 100 Watt power, the incident irradiance expected at the first surface of the thermochromic layer is approximately $I_{inc} \approx 58 \text{ kW/m}^2$.

The thermal diffusivity of the thermochromic glass is poor

$$\delta = \frac{\kappa}{\rho c_p} \approx 5 \times 10^{-7} \text{ m}^2/\text{sec} \quad (102)$$

so the thermal diffusion time constant for a 1 cm^2 patch is approximately 200 sec. The surrounding polymer has an even lower thermal conductivity, so it is not expected to significantly impact the thermal dissipation. With a slow thermal relaxation rate, we expect the pump laser radiation absorbed in the thermochromic layer to raise the temperature on-axis at an initial rate of approximately

$$\frac{dT}{dt} \approx \frac{I_{inc} \alpha}{\rho c_p} \approx 8^\circ \text{C/sec} \quad (103)$$

As the temperature in the thermochromic layer increases, thermal diffusion will compete and the temperature rate will decrease. If we are interested in observing damage in the thermochromic layer, the temperature will have to rise from 20°C ambient by about 40°C . The crude analysis above indicates that the time scale of the simulation will be of order several seconds, and that a good plotting interval may be 1-2 seconds.

C.5 Configuration Parameters

During initial investigation of a simulation scenario, it is beneficial to ease the constraints on the radiation dose, timestep, and the maximum allowed solver error in order to make the simulation run faster. The idea is to make sure that the temporal dynamics of the simulation match the expected behavior prior to investing a great deal of time to simulate the scenario accurately. The configuration file below has settings that are appropriate for a first look at a problem. The radiation step has a relatively small, manually set number of rays (10000), the SOR convergence tolerance is 0.05 °C, and the global temperature error tolerance is 0.1 °C. The maximum timestep is set equal to the expected time constant for the dynamic we are investigating.

```
1 | # thermochromic.yaml
   | #
   | # Simulate thermochromic behavior in a doped glass layer in a polymer sphere
5 | # simulation parameters
   | object          : "object/sphere-layer.obj"
   | material_lib    : "material/lib.yaml"
   | material_dir    : "material/"
   | output_prefix   : "output/"
10 |
   | # radiation params
   | ray_weight_thresh : 0.001
   | num_rays          : 10000
   | smart_dose_thresh : 0.1
15 |
   | # diffusion params
   | ambient_temp      : 20          # deg C
   | initial_temp      : 20          # deg C
   | sor_omega         : 1.6
20 | sor_converge_tol   : 0.05        # deg C
   |
   | # thermal boundary conditions
   | bc_rad           : off
   | bc_convect       : off
25 | bc_evap           : off
   |
   | # perfusion (coolant) parameters
   |
   | # time step parameters
30 | timestep_max       : 1.0          # sec
   | error_tol         : 0.1          # deg C
   |
   | ---
   | #####
35 | # define aliases to be used during simulation
   |
   | # define a powerful pump laser source (SWIR)
   | # tophat source aimed in z-direction at center of sphere
   | - source: &pump { id: 0 }
40 |   type           : tophat
   |   wavelength    : 1370          # nm
   |   power         : 0.0           # Watts
```

```

radius      : [0.05, 0.05]      # m
center      : [ 0.12, 0.12, -1.0 ]
45 target    : [ 0.12, 0.12, 0.12 ] #aim at center of sphere
quasirandom: true

# define a weak probe laser source (VIS)
# tophat source aimed in -z-direction at center of sphere
50 # counter-propagate the pump laser
- source: &probe { id: 1 }
  type      : tophat
  wavelength : 500          # nm
  power     : 0.0           # Watts
55 radius    : [0.01, 0.01]      # m
  center    : [ 0.12, 0.12, 1.0 ]
  target    : [ 0.12, 0.12, 0.12 ] #aim at center of sphere
  quasirandom: true

60 #define an alias that will pump the thermochromic material
- &pump_it
  - source: *pump
    power: 100.0
  - source: *probe
65   power: 0.0

#define an alias that will probe the thermochromic material
- &probe_it
  - source: *pump
70   power: 0.0
  - source: *probe
    power: 0.01

# define an alias to output the axial
75 # temperature and damage fields, along z-axis
- &plot_axial
  - plot: temperature
    prefix: "axial"
    x: [0.12, 0.12]
80    y: [0.12, 0.12]
  - plot: damage
    prefix: "axial"
    x: [0.12, 0.12]
85    y: [0.12, 0.12]

# define an alias to output the axial
# pump laser dose, along z-axis
- &plot_pump
  - plot: dose
90   prefix: "pump"
    x: [0.12, 0.12]
    y: [0.12, 0.12]

# define an alias to output the axial
95 # probe laser dose, along z-axis
- &plot_probe

```

```

100   - plot: dose
      prefix: "probe"
      x: [0.12, 0.12]
      y: [0.12, 0.12]

# define an alias to output the full
# pump laser dose
105   - &plot_full_pump
      - plot: dose
        prefix: "full_pump"

# define an alias to output the full
# temperature field
110   - &plot_full_temp
      - plot: temperature
        prefix: "full_temp"

#####
115 # START THE SIMULATION
    - start

    - repeat: 10
      do:
120         - *probe_it                # turn on the probe beam
          - update_dose
          - *plot_probe                # plot the probe dose
          - *pump_it                  # turn on the pump beam
          - update_dose
125         - *plot_pump                # plot the pump dose
          - wait: 2.0                 # it gets absorbed in the layer
          - *plot_axial                # plot the temperature and damage

130   - *plot_full_pump                # full 3D pump dose field
      - *plot_full_temp                # full 3D final temperature field

      - *probe_it                    # final probe
      - update_dose
      - *plot_probe
135   - stop
# END OF THE SIMULATION

```

C.6 Action Script Aliases

Using YAML aliases is a good way to abstract the details of the action script, so that the sequence of actions after simulation start is clear and understandable. Aliases are defined in the action script prior to the `start` command, and are typically used to set up radiation sources, plot file output, and more complex action sequences. In this simulation, we will use two radiation sources, defined and aliased, one for the pump laser and one for the probe laser. They are both collimated top-hat beams, but having different beam radii, wavelengths, and powers. The pump and probe lasers are set to be counter-propagating and coaxial, aligned parallel to the z-axis of the coordinate system, and passing through the center of the object.

Next, aliases (`pump_it` and `probe_it`) are defined that turn the pump laser on and the probe laser off, and vice versa. These aliases work by simply changing the `power` parameter of each source. Following are a number of plotting aliases for saving temperature, damage, and dose data to files for post-analysis. All of the physics is happening along the central beam axis, so 1D plots of relevant quantities are sufficient for analysis. It is useful to also include a full domain plot (like `plot_full_temp`) to make sure that there is nothing unexpected happening in other parts of the computational domain.

C.7 Action Script Simulation Timeline

The end of the action script has the actual commands that control the simulation timeline, occurring after the `start` command. The simulation will begin with a measurement of the probe beam absorption (instantaneous in time), followed by a 2 sec exposure of the object to the pump beam. These steps repeat 10 times for a total simulation duration of 20 seconds. At the end of the simulation, the full domain temperature and pump absorption are saved, along with a final probe absorption measurement.

Its now time to run the simulation. From the simulation directory, we run SESE on our workstation, using four computational threads:

```
|| sese -t4 config/sim-thermochromic.yaml
```

C.8 Simulation Results and Visualization

It is good organizational practice to keep the routines and data analysis products in a separate directory, called, of course, `analysis`. The first thing we will inspect is the full temperature field at 20 sec simulation time (the end of the run). We expect to see that most of the temperature rise is in the thermochromic glass layer, with possibly a small temperature rise in the polycarbonate sphere. The first step is to convert the `full_temperature` file to xdmf format. From the simulation directory, the command

```
|| raw3dtoxdmf output/full*.out
```

converts the temperature and the pump dose files to the xdmf format. Now launch Paraview and open the file `full_temp_temperature_20.0000000000.out.xmf` using the XDMF reader. Press the Apply button to perform the read. Next, choose the slice tool and make a slice plane with a Y normal vector and positioned at the origin. Orient the camera to view the object with the z-axis up and the x-axis to the right. Press Apply to obtain the visualization in Figure C-3.

The pump beam enters the object from the $-z$ direction, and there is evidence that the pump beam is refracted by the spherical surface of the polycarbonate sphere, and that there is a small amount of absorption and thus temperature increase in the polycarbonate material. The filamentary appearance of the beam in the polycarbonate is due to the discretization of the PLIC surface of the sphere - a finer discretization would reduce these artifacts. We observe that the absorption is so strong in the thermochromic layer that there is no evidence of the pump beam on the $+z$ side of the thermochromic layer. From the plot, the peak temperature at the end of the simulation is above 60 °C, so we anticipate that the thermochromic material will have sustained significant damage in the central region of the layer.

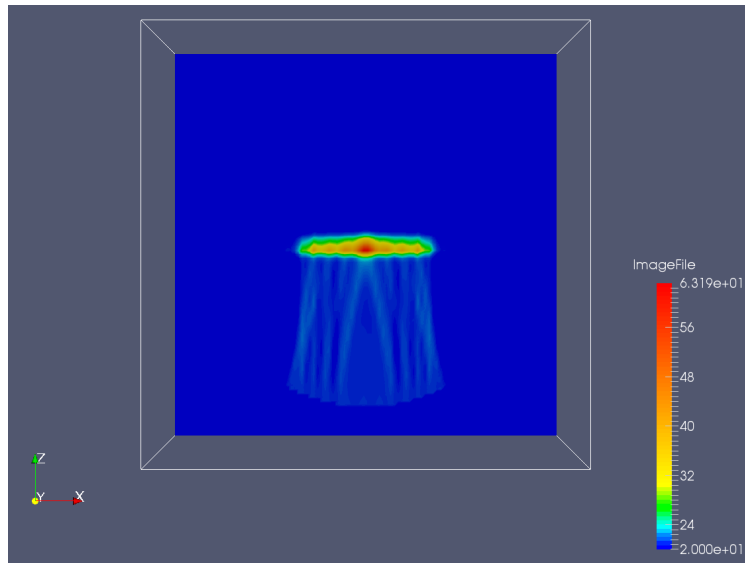


Figure C-3. ParaView visualization of slice through full temperature field

Paraview visualizes 3D and 2D data. The best way to analyze the 1D axial data files produced by this simulation is with a more procedural environment like Python, MatLab, or IDL. Here we will show how to use the GNU Data Language (GDL) to process and visualize the axial data. GDL is a free, open-source version of Harris/Exelis Interactive Data Language (IDL), which is a powerful tool for doing data processing and visualization. A GDL procedure (.pro file) called `sese_read.pro` has been included in the SESE distribution for directly reading the RAW3D file format produced by SESE. Thus, no conversion to text or XDMF format is required when using GDL.

The following GDL commands will read the initial (time=0) axial probe beam dose and plot it in an X11 window:

```
fn = 'output/probe_dose_0.0000000000.out' ;name of file
to read
data = sese_read(fn) ;read the file
into a structure
zaxis = indgen(data.nz) * data.hz - data.nz * data.hz / 2.0 ;make vector
with z axis values
plot, zaxis, data.mesh, xtitle='Z (m)', ytitle='Dose (W/m^3)' ;plot the axial
dose profile
```

A slightly more complicated set of commands will procedurally read the time series of probe dose files into an array and will then plot the time=0 sec and time=20 sec doses on the same axes, with the y-axis range clipped to a value of 100:

```
1 | timev = indgen(11) * 2.0D0 ;make a vector
of times to analyze, 0..20
ntime = n_elements(timev) ;number of
times to analyze
;
```

```

5 ; read in probe dose files and plot
;
for j = 0L, ntime-1 do begin $                                ;loop through
    times
    ts = strtrim(string(timev[j], format='%"-20.10f"')) & $ ;make a
        timestamp string
    fn = 'output/probe_dose_' + ts + '.out' & $                ;construct the
        file name
    data = sese_read(fn) & $                                    ;read in raw3D
        file
10    if j eq 0 then begin $                                    ;set up array
        on first pass
        probe = dblarr(data.nz, ntime) & $                    ;allocate array
            of doubles
        endif & $
        probe[:, j] = reform(data.mesh) & $                    ;insert data
            into array
    endfor
15 ;
plot, zaxis, probe[:,10]<100, $
    xtitle='Z (m)', ytitle='Probe Dose (W/m^3)', charsize=2, thick=2
oplot, zaxis, probe[:,0]<100, thick=2, linestyle=2            ;plot the first
    and last axial probe dose
;
20 print, 'Peak Probe Dose at t=20 sec = ', max(probe[:,10])

```

The resulting plot is shown in Figure C-4. The probe laser beam originates from a positive z-axis position, and travels through the origin toward negative z values. It is apparent from the dose plot that the dose is larger on the +z side of the plot, indicating that some absorption is taking place. In addition, the absorption, and thus dose, is clearly higher in the thermochromic glass layer at the center of the object. We also note that the absorbed probe beam dose is larger at the later time, indicating that the temperature of the thermochromic layer increased by pump beam absorption. The peak probe dose at t=20 sec is about 1346 W/m³, indicating that the probe beam absorption is about 22 times larger at the later time/higher temperature. This is very close to what we would expect from the temperature dependence of the thermochromic material, as specified in the material file. Finally, the probe laser dose plot indicates that probe absorption has decreased below the initial t=0 value on the -z face of the thermochromic layer. We surmise that since the pump laser beam travels from -z to +z, there has been enough absorption and heating of the surface of the thermochromic layer facing the pump beam to cause damage, irreversibly destroying the chromophores in the glass and causing the probe absorption in that immediate region to be reduced to a very low level.

As verification that there was temperature-induced damage, the temperature and accumulated damage are plotted at location ($x=0, y=0, z=-0.0048$), as a function of time. This location coincides with the location of the peak temperature in the simulation. These plots are shown in Figure C-5 and Figure C-6, respectively. The temperature plot indicates that the initial temperature rate is approximately 5 °C/sec, which is close to the estimated value, and that the peak temperature in the thermochromic glass layer exceeds 50 °C, where chromophore damage rapidly takes place. Further indication of chromophore damage comes from Figure C-6, where the log of the cumulative

damage exceeds a value of 0 (log of 1.0) at about 10 sec into the simulation. At this time, the peak temperature is about 48 °C, a temperature at which the damage rate is increasing rapidly.

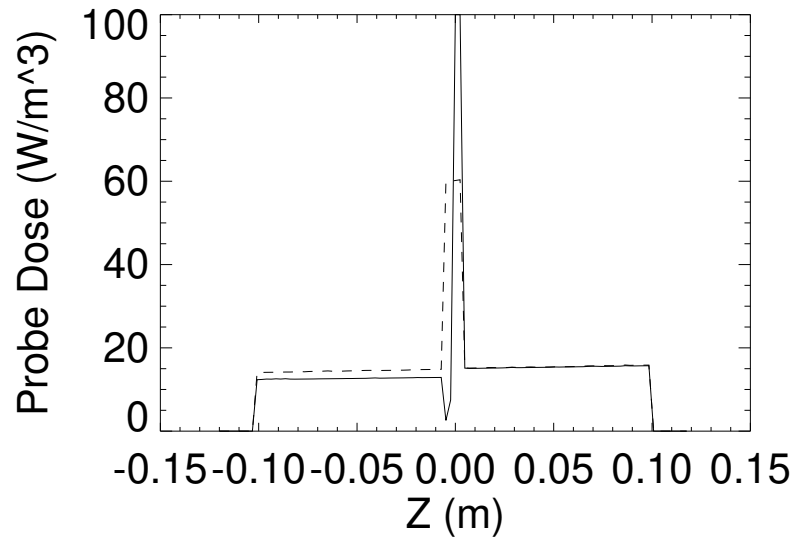


Figure C-4. Comparison of probe laser axial dose at time 0 sec (dashed) and 20 sec (solid)

In this section, we have walked through the process of setting up a complete SESE simulation, including directory organization, material files, configuration settings, and action script. Even though the simulation was of a somewhat contrived scenario, the physics, the setup, and the analysis of the simulation results is relevant to real problems that SESE simulates.

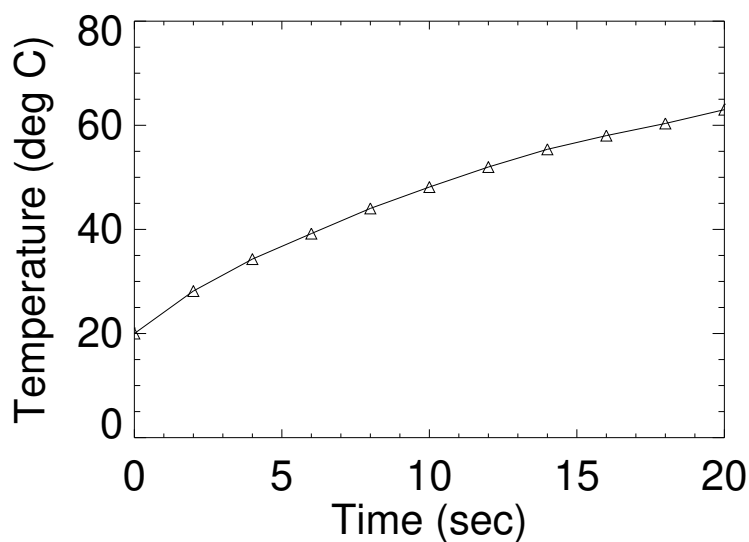


Figure C-5. Temperature at $-z$ surface of thermochromic layer, as a function of time

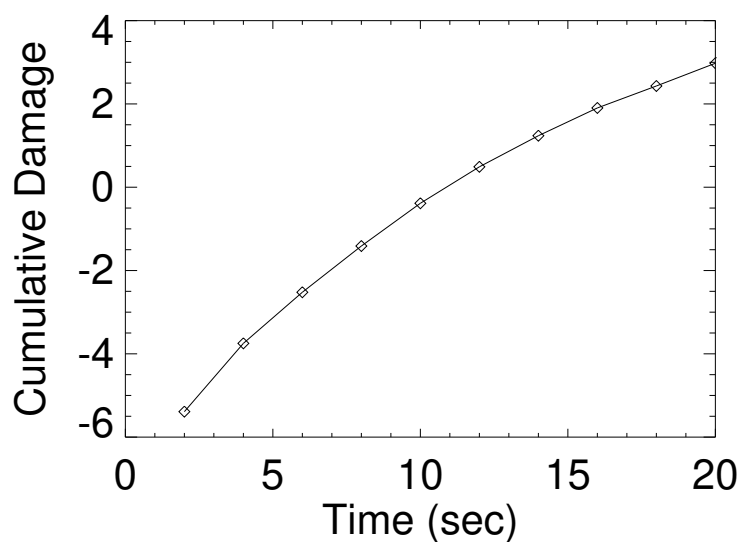


Figure C-6. Log of cumulative damage at $-z$ surface of thermochromic layer, as a function of time

C.9 Strategies For Numerical Simulation With SESE

SESE attempts to simulate the interaction of electromagnetic radiation with materials by solving a set of coupled partial differential equations in space and time, with inhomogeneous coefficients, in a complex and temporally dynamic computational domain, with versatile boundary conditions, and with high spatial and temporal fidelity. Depending on the complexity of the problem, performing

the simulation may take a long time, even when SESE is running in a high-performance computational environment. This section tries to assist with how to approach computational simulation with SESE, and presents strategies for maximizing success.

As with most complex problems, it is always best to start with the simplest case that is practical and meaningful and add complexity from there.

C.10 Exploit Symmetry

If your problem has symmetry, then computational run times and memory requirements can usually be reduced by large factors. For example, if your object has bilateral symmetry, placing the symmetry plane along a boundary of the computational mesh imposes an automatic zero-flux boundary condition on the symmetry plane, which is consistent with the absence of heat flow across symmetry planes. Radiation sources that propagate within symmetry planes do have some fraction of their rays wasted because they do not interact with the object, but this usually incurs only a very small time penalty compared with the savings of a much-reduced computational domain. In future releases, SESE will include the option of selecting symmetry planes in radiation sources that will eliminate the wasted-ray penalty.

C.11 Isolate Radiation Sources

If your SESE run includes multiple radiation sources, or complex movement of radiation sources, check correct configuration of the sources prior to running with the heat equation solver. This can be done by temporarily filling your object with a weakly absorbing "test material", and then configuring SESE to output the radiative dose. Visualizing the location of the dose relative to the features in the object will let you verify the desired position and pointing of the source. This technique can also be used to verify that the focal parameters and irradiance distribution of the sources are what you were expecting.

C.12 Distribute The Load Wisely

Running large simulations successfully in a high-performance computational environment can sometimes be challenging. Often, there is little insight about the actual resources required by a program, and there can be penalties for over- or under- utilizing the available computer hardware resources. SESE's requirements for CPU and memory can be broken into two phases: the radiative step and the heat step. Broadly speaking, each MPI node used in a run will load some fraction of the entire object into its local memory. The fraction that is loaded, and the resulting memory required, depends on the extent of the radiative dose. If the source radiation is limited to a small region of the object and/or the absorption depth is small, then the amount of memory required per MPI node is small in the radiative step. If a large fraction of the object is illuminated, or if radiation can penetrate the object deeply, then memory requirements per node will grow. During the radiative step all threads in an MPI node have access to the node's memory, and all threads propagate threads in parallel. We conclude that for the radiative step, it is beneficial to have a large number of MPI nodes, and to have each MPI node use the largest number of threads possible for the given hardware configuration.

In the heat step, the object is broken evenly into slabs along its z axis, one per MPI node. Solution of the heat equation requires a significant amount of communication among the MPI nodes, and

the size of messages is proportional to the number of voxels in the x and y directions. Therefore, it is usually best to orient the object with its long dimension to be along the z axis to minimize the MPI message size. In addition, consider balancing the amount of computation and messaging per MPI node. In a given MPI node, threads all work on the solution of the heat equation, and then only a single thread runs to send and receive MPI messages. If too many MPI nodes are used, then the slab size for a node will be small - the heat equation solver will finish its computation quickly and messaging dominates the time taken for the heat step. The optimum number of MPI nodes depends on the speed of the inter-node communications backbone, and the amount of memory and the number of threads supported by each node, and typically needs to be found empirically for a given HPC system.

C.13 Start With Low Fidelity

Begin your simulation with a coarse voxelization of your object, large temperature error tolerances, and a large radiation dose uncertainty. If you know that your ultimate simulation will take a long time to run, you may even wish to consider starting with a 2D or even a 1D simulation. Using a lower dimensionality and/or a coarse voxelization may provide insight regarding the time scale of important phenomena.

If you are using the `makegeom` routines to generate your object, changing to a coarse voxelization or a lower dimensionality can be as simple as changing the values of the `set_dims(nx, ny, nz)` member function to vary the level of discretization. Use large values (~ 0.4) of the `rad_uncertainty_tol` parameter to increase the speed of the radiation step. Similarly, setting a large value (~ 0.3) for the `smart_dose_thresh` parameter will tell SESE to re-use the radiative dose unless major changes occur to the material optical parameters. Setting the `sor_converge_tol` and `error_tol` parameters to ~ 0.2 and ~ 0.5 , respectively, will speed the heat diffusion calculations. Finally, unless your simulation critically needs it, turn off the `internal_boundary_refract` parameter to speed the radiative step. With these low fidelity settings, SESE should perform computations relatively rapidly, and you can monitor intermediate results and/or the final result to make sure that the simulation is running as expected.

C.14 Do a Sanity Check

Look carefully at the results from the low fidelity simulations to make sure that 1) SESE is simulating the scenario you intend it to; 2) the results make sense, and are reasonable for the physics being simulated; 3) the expected physical phenomena are occurring at the right time and place and under the right conditions; 4) the simulation is advancing without unreasonably small time steps during less dynamic parts of the run.

Be sure to look at the accumulated damage field to make sure that the object is changing its parameters as you expect it to. Inspection of the radiative dose at various times can also provide insight on how material optical parameters are changing.

C.15 Check Scaling

If you are confident that the low-fidelity results look good, then it is time to start advancing toward the full high-fidelity simulation. First, increase the fidelity of the simulation by changing the `rad_uncertainty_tol`, `smart_dose_thresh`, `internal_boundary_refract`

and timestep limits to values that are more consistent with a high-fidelity simulation, while keeping the spatial resolution of the computational mesh relatively coarse. These changes will probably increase the run time significantly, but it should still be manageable for the purposes of runtime scaling.

Next, the best way to estimate the time that the full simulation will take is to increase the total number of computational mesh points by a convenient factor, say 2, and observe the effect on the simulation run time. The scaling is not linear, so this doubling process should be repeated to obtain a reasonably accurate estimate. Make sure that your expectations on the total high-fidelity simulation runtime estimate are reasonable. At this point, it is typical to make a tradeoff between spatial resolution and runtime in order to get the most acceptable results in the fastest time frame. Only you (or your boss) can make that determination.

C.16 Go For It

It is now time for the full simulation. Double check your configuration files, make sure that you have adequate disk space for all those big plot files, and be sure that there are no scheduled computer down times. Start the simulation and monitor intermediate results until you are satisfied that the simulation is proceeding as expected. Intermediate data output files, such as radiation dose plot files, temperature plot files, and volfrac plot files, are available as soon as they are written by SESE. A good simulation action script will output these intermediate quantities periodically during the run. If disk space is an issue, specify a subdomain, for example, a 2D slice through a plane of symmetry. Using MATLAB, GDL (IDL), or Paraview, open these files and assess whether the “large” simulation run is progressing as expected, given the results of your previous “small” simulation runs. In addition, SESE prints solver status information, with convergence criteria, solver CPU time expended during the various steps of the calculation, and determination of the limiting factor for the simulation timestep. This ongoing stream of console output can be used to estimate simulation completion time by extrapolating the wall clock time vs. simulation time.

C.17 Visualization Of Results

In this section, we walk through the procedure for visualizing the output of a SESE simulation run. Specifically, we will be using ParaView to inspect, visualize, and plot the temperature field from the "sim-onepulse" example contained in the SESE distribution. ParaView is a tool for visualizing two- and three- dimensional data sets, capable of processing very large data sets with the use of multi-processor distributed memory supercomputers or workstation clusters. The full ParaView application usage and abilities is not within the scope of this document. For additional information regarding the ParaView program, please visit www.paraview.org and read through the wiki and tutorial pages.

In the "sim-onepulse" example, the action script creates five temperature field output files: a full object distribution, a longitudinal slice through the object center, a transverse slice at the object surface, a transverse slice 1mm into the object surface, and an axial temperature distribution. The default "output_prefix" parameter in the configuration file sets the file system location that these output files are created upon execution of the simulation. Each of the simulation output files (*.out file extension) is written in a binary format called "raw3D", and must be converted into a format compatible with ParaView. Several file format conversion scripts are included in the SESE

distribution under the "bin" directory. For this example, the `raw3dtoxdmf` executable is used to convert SESE *.out files to *.raw and *.xdmf files of the same name as the original SESE output file. These files are readable by ParaView. Typing

```
cd output
raw3dtoxdmf *.out
```

at the command prompt converts all the SESE output files within the current directory. For the purposes of this example, this method converted all SESE output files.

ParaView and all of its dependencies are assumed to have been previously installed. Open ParaView and select the "Open" option from within the "File" menu (Ctrl+O). In the open file dialog box, navigate through the file system to the directory containing the converted SESE output files. Similar file names automatically group the converted files, as shown in Figure C-7.

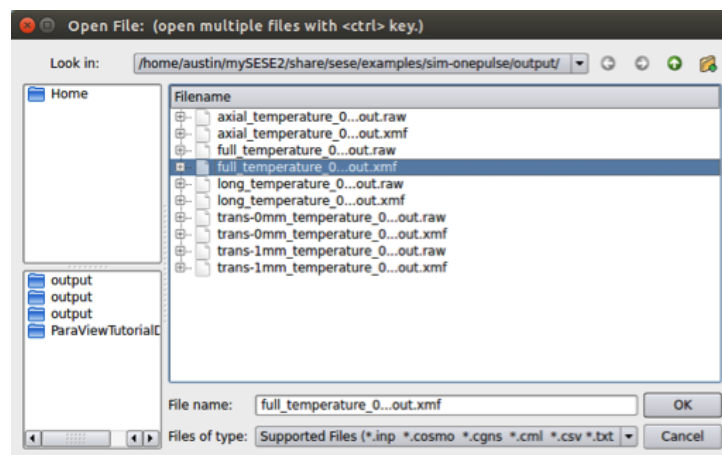


Figure C-7. ParaView open file dialog box with converted SESE output

Each file group can be expanded to select individual files, or the group can be selected as a whole by selecting the file group name. Multiple output files and groups can be selected (on a PC) with the Ctrl key. For this example, the full temperature data set is chosen. Clicking "OK" prompts for the proper data reader. Xdmf Reader should be selected and clicking "OK" again reads the dataset into the ParaView pipeline browser. This process is depicted in Figure C-8

Pressing the green "Apply" button within the Properties toolbar plots the dataset within the layout window. At this point, parameters are set to customize the way the interpretation and display of data within the rendered layout. In the "Active Variable Controls", "Solid Color" is changed to "ImageFile". In the "Representation Toolbar", "Outline" is changed to "Surface".

Using the presets in the "Camera Control" toolbar or by clicking and repositioning the rendering within the layout window, reposition the dataset object. Using the "VCR Controls" the dataset representation can scan through the available time steps in the simulation output, visualizing the temporal output of the simulation, as shown in Figure C-10.

To inspect the temperature data more closely, use the "Representation Toolbar" and select "Slice"

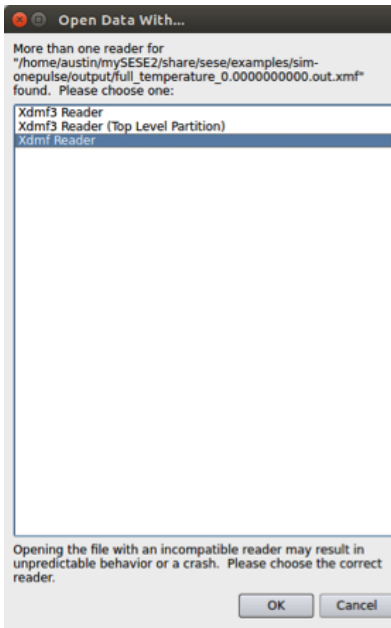


Figure C-8. Input file reader selection menu

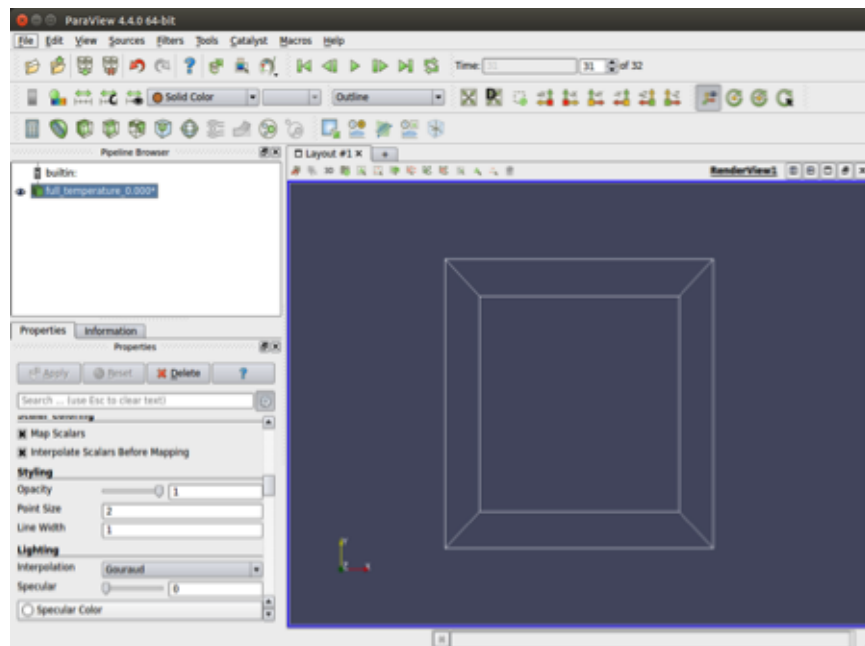


Figure C-9. Data set display setup

to display a 2-D slice through the 3D dataset. Under the "full_temperature_0.000*" properties menu, the slicing direction and slice position can be selected. When viewing the slice, the time dependence of the data can also be visualized using the "VCR Controls", as shown in Figure C-11

There are many ways of displaying the 3-D and 2-D information of data sets within ParaView. The

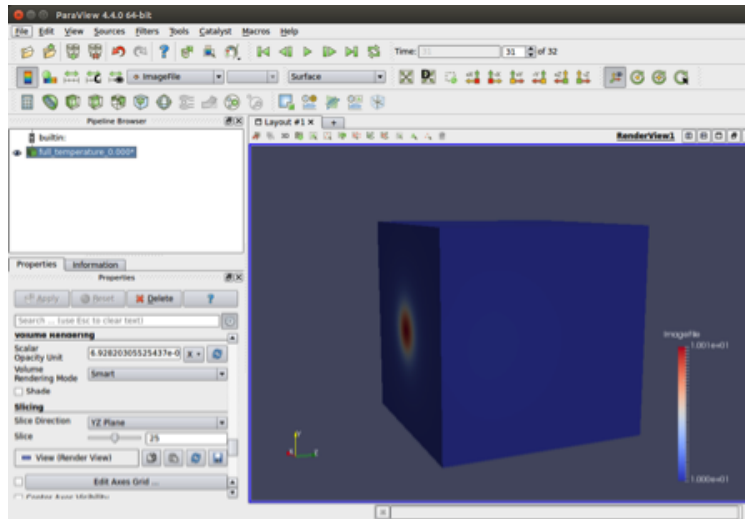


Figure C-10. Object orientation and transient display

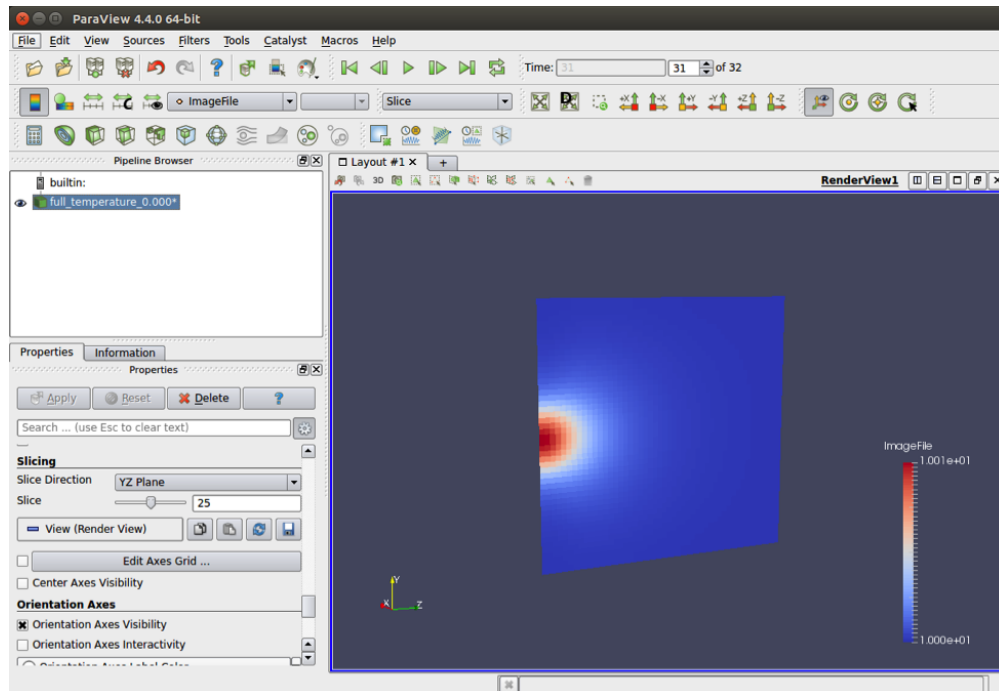


Figure C-11. 2D slice rendering

program is complex, and sometimes a little quirky, and becoming proficient in its use requires a thorough understanding of the documentation. To visualize the multiple slices of temperature field output, like those shown in Figure C-12, each individual 2-D slice dataset must be opened and rendered. The dataset only contains information within the two dimensions of recorded data. In order to visualize these slices together, Paraview must configure the inner slices with the appropriate translation.

For the "sim-onepulse" example: "long_temperature_0.000*" shows a transient XZ slice at 1mm in Y from origin, so the dataset is translated 0.001 meters in Y; "trans-0mm_temperature_0.000*" shows transient XY slice at 0mm in Z from origin and requires no translation; "trans-1mm_temperature_0.000*" shows transient XY slice at 1mm in Z from origin and is translated 0.001 meters in X. Visualizations can be exported to most popular image file formats for inclusion in reports or presentations.

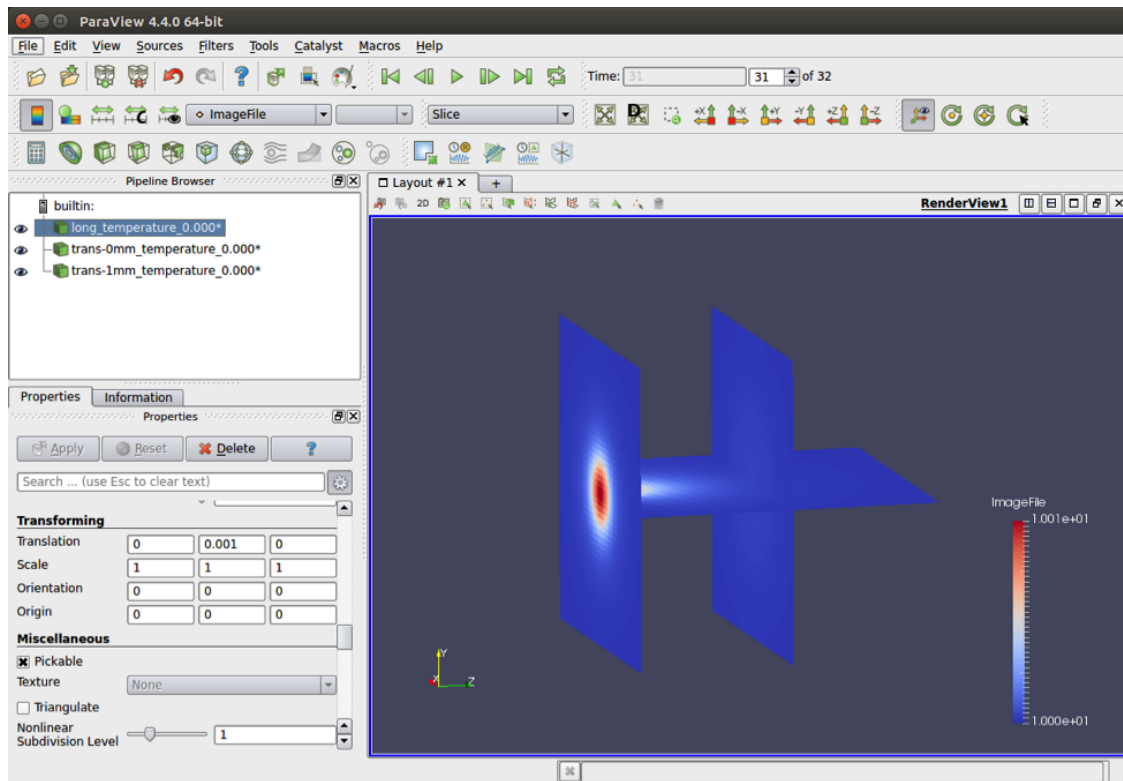


Figure C-12. Visualization of multiple 2D slices of a 3D dataset

APPENDIX D - KEYWORD REFERENCE

D.1 Configuration File Keywords

Table D-1. SESE configuration file parameters appearing in the first YAML document in the configuration file. Parameters marked mandatory must appear in the configuration file

Parameter	Description
object	(string) mandatory path to object file
material_lib	(string) mandatory path to materials library
material_dir	(string) mandatory path to directory containing material files named in materials library
one of num_rays or rad_uncertainty_tol	mandatory in the Monte Carlo radiative transfer simulation, propagate rays either until num_rays rays have been propagated or the uncertainty in the dose distribution is less than rad_uncertainty_tol
ambient_temp	(°C) mandatory set the temperature uniform throughout exterior domain (where material ID is zero)
initial_temp	(°C) mandatory set the initial temperature uniform throughout interior domain (where material ID is nonzero)
output_prefix	(string) prefix to apply to output files; may contain path to an output directory (default: " ")
object_volfrac	(string) path to object's volume fraction file (default: " ")
iso_scatter_thresh	in the Monte Carlo radiative transfer simulation, trigger effective isotropic scattering with reduced scattering coefficient in media where $\alpha/\sigma(1 - g)$ is below this value (closer to 0 $\Rightarrow$ more accurate; default: 1e-3)
ray_weight_thresh	in the Monte Carlo radiative transfer simulation, terminate propagation of a ray when this fraction of its energy has been absorbed (closer to 0 $\Rightarrow$ more accurate; default: 1e-3)
roi_size	number of high-energy voxels to select as a region of interest for the Monte Carlo radiative transport uncertainty calculation (default: 100)
smart_dose_thresh	(float) threshold relative change in any optical quantity that triggers a recomputation of the dose distribution when exceeded. default: -1.0 (i.e. recompute every cycle)
smart_dose_dt	(float) maximum time interval that may elapse before triggering a recomputation of the dose distribution. default: 1e20
sor_omega	in the thermal diffusion simulation, use this over-relaxation parameter to accelerate the red-black SOR method (usually somewhere between 1 and 2; increase to yield fastest convergence without sacrificing stability; default: 1.0)
Continued on next page...	

Table D-1 – continued from previous page

Parameter	Description
<code>sor_converge_tol</code>	(°C) in the thermal diffusion simulation, terminate the red-black SOR iterative method when the temperature change per iteration falls below this value (must be smaller than <code>error_tol</code> ; default: $1e-6$)
<code>error_tol</code>	(°C) adaptively adjust the thermal diffusion time step to limit error in temperature to this tolerance in every step (must be larger than <code>sor_converge_tol</code> ; default: $1e-3$)
<code>timestep_min</code>	minimum time step allowed; the time step may drop below than this value only if SOR iteration fails, or if surface recession per time step exceeds the threshold set by <code>max_recession_per_step</code> (default: 0.0)
<code>timestep_max</code>	maximum time step allowed (default: $1e20$)
<code>timestep_ratio_min</code>	minimum factor by which to reduce adaptive time step (default: 0.0)
<code>timestep_ratio_max</code>	maximum factor by which to increase adaptive time step (default: 2.0)
<code>pool_volume</code>	(m^3) total effective volume of perfusion reservoir (default: 0.0 , i.e. perfusion disabled)
<code>pool_vhc</code>	($J/m^3 \cdot K$) volumetric heat capacity of perfusion reservoir (default: 0.0)
<code>pool_temp</code>	(°C) initial temperature of perfusion reservoir; if <code>pool_volume</code> is finite and non-zero, the pool temperature will be updated over time to conserve energy as heat is exchanged between the object interior and the pool (required if <code>pool_volume</code> > 0.0)
<code>bc_rad</code>	(on/off) apply radiative boundary condition (default: <code>off</code>)
<code>bc_convect</code>	(on/off) apply convective boundary condition (default: <code>off</code>)
<code>bc_evap</code>	(on/off) apply evaporative boundary condition (default: <code>off</code>)
<code>rel_humidity</code>	(float) ambient relative humidity in $[0, 1]$ (required if <code>bc_evap</code> is present)
<code>gaussian_beam_step</code>	(float) Gaussian beam trajectory step size, as a fraction of the radius of curvature hyperbolic Gaussian beam trajectories. default: $1e-4$
<code>internal_boundary_refract</code>	(true/false) enable reflection/refraction at internal material interfaces. default: <code>false</code>
<code>internal_refract</code>	(true/false) enable internal reflection/refraction at <i>any</i> refractive index change (including refractive index gradients). default: <code>false</code>

D.2 YAML Action Script Keywords

D.3 Radiation Source Keywords

Table D-2. Keys passed with `source` key, applying to all source types. While not all parameters need to be given with every appearance of the `source` key, all necessary parameters must be specified at least once before the simulation begins.

Key	Value
<code>source</code>	(map) a YAML map containing the <code>id</code> key, identifying the source to which the other keys apply
<code>id</code>	(integer, mandatory) number uniquely identifying the source
<code>type</code>	(string) the source type; may be one of <code>point</code> , <code>tophat</code> , <code>gaussian_beam</code> , <code>file</code> ; must be present in the first appearance of each new source definition
<code>wavelength</code>	(integer, optional) wavelength tag of source, corresponding to set of optical properties in material files
<code>power</code>	(float) set the source power [W]
<code>center</code>	(sequence of three floats) set the center of the source grid at the given point [m]
<code>target</code>	(sequence of three floats) ‘aim’ the source target at the given point [m]
<code>twist</code>	(float) ‘twist’ the source plane about its axis by the given angle (for use with <code>target</code> only) [degrees]
<code>euler_angles</code>	(sequence of three floats) orient the source using the given $ZX'Z''$ intrinsic Euler angle rotation, starting with the source plane aligned with the object <code>xyz</code> coordinate system [degrees]
<code>radius</code>	(sequence of two floats) independent radii of a primitive source distribution; for <code>gaussian_beam</code> source type, this sets $1/e^2$ radii [m]
<code>radius_curvature</code>	(sequence of two floats) independent radii of curvature of a primitive source distribution. default: $1e20$ [m]
<code>quasirandom</code>	(true/false) for top-hat and Gaussian beam source types, use quasirandom sampling to speed up Monte Carlo convergence. default: <code>false</code>
<code>spread</code>	(string) random angular spread, one of <code>cone</code> , <code>cosine</code> . does not work with source type <code>gaussian_beam</code>
<code>spread_cone_angles</code>	(sequence of two floats) with <code>spread:cone</code> , emit rays uniformly within the given range of angles. default: <code>[0,0]</code> [degrees]
<code>spread_cosine_exponent</code>	(float) with <code>spread:cosine</code> , emit rays with polar angle θ according to the distribution $\cos^n \theta$, where n is the given value. default: <code>1</code>
<code>tophat_obscure</code>	(sequence of two floats) offset of elliptical obscuration from source center. default: <code>[0,0]</code> [m]
Continued on next page...	

Table D-2 – continued from previous page

Key	Value
tophat_obscure_radius	(sequence of two floats) radii of elliptical obscuration. default: [0,0] [m]
file	(string, required if type=file) path to the source file relative to working directory

D.4 Plot Output Keywords

Table D-3. Keys passed with the plot key

Key	Value
plot	(string, required) the name of the field to plot: temperature, damage, dose, volfrac, surface, speed, normal_x, normal_y, normal_z
prefix	(string, optional) prefix to the output file; may contain a path to another directory
x	(2-tuple, optional) restrict plotting of the x coordinate to the given range in [meters]
y	(2-tuple, optional) restrict plotting of the y coordinate to the given range in [meters]
z	(2-tuple, optional) restrict plotting of the z coordinate to the given range in [meters]

D.5 Repeat Keywords

Table D-4. Keys passed with the repeat key

Key	Value
repeat	(integer, required) number of times to repeat an action or sequence of actions
do	(YAML node, required) an action or sequence of actions to repeat; may be a YAML scalar, map or sequence, or nested maps and sequences

D.6 Material File Keywords

Table D-5. Definitions of keywords used in material files to specify material properties

Parameter	Units	Dependence	Description
optical			a sequence of YAML maps, one for each wavelength, each containing optical parameters for the material
wavelength	nm	T, f, λ	specifies the valid wavelength for the optical properties appearing in the same YAML map (default=0)
refractive_index		T, f, λ	index of refraction (default=1.0)
absorption	m^{-1}	T, f, λ	linear optical absorption coefficient (default=0.0)
scattering	m^{-1}	T, f, λ	linear optical scattering coefficient (default=0.0)
scattering_g		T, f, λ	Henyey-Greenstein optical scattering anisotropy factor, $-1 \leq g < 0$: backward scatter, $g = 0$: isotropic, $0 < g \leq 1$: forward scatter, (default=0.0)
thermal			a single YAML map, containing thermal parameters for the material
density	kg/m^3	T, f	density of the material (default=1.0)
conductivity	$\text{W}/(\text{m} \cdot \text{K})$	T, f	thermal conductivity (default=0.0)
specific_heat	$\text{J}/(\text{kg} \cdot \text{K})$	T, f	specific heat capacity (default=1.0)
metabolic_rate	W/m^3	T, f	internal metabolic heat generation (default=0.0)
perfusion	s^{-1}	T, f	perfusion rate - volumetric flow rate of perfusant per voxel volume (default=0.0)
heat_transfer_coeff	$\text{W}/(\text{m}^2 \cdot \text{K})$	T, f	convective heat transfer coefficient (default=0.0)
evap_coeff	$\text{W}/(\text{m}^2 \cdot \text{Pa})$	T, f	evaporative heat transfer coefficient (default=0.0)
emissivity		T, f	surface radiative emissivity (default=0)
chemical			a single YAML map, containing chemical reaction parameters for the material
arrhenius_prefactor	s^{-1}	T	Arrhenius rate coefficient for chemical damage (analogous to collision rate) (default=0.0)

activation_energy	J/mol	T	molar activation energy for the chemical damage reaction (default=1.0E20)
-------------------	-------	-----	---

D.7 Geometric Primitives For makegeom

Table D-6. Geometric primitive classes provided for object generation. All arguments have double type and units of meters

Primitive Class	Arguments	Description
Sphere	(x, y, z, r)	Draw a sphere centered at (x, y, z) with radius r .
Halfspace	(x, y, z, n_x, n_y, n_z)	Draw a halfspace at point (x, y, z) with outward normal vector (n_x, n_y, n_z) .
Cylinder	$(x, y, z, a_x, a_y, a_z, l)$	Draw a cylinder of length l centered at (x, y, z) with axis vector (a_x, a_y, a_z) .
Box	$(x_1, y_1, z_1, x_2, y_2, z_2)$	Draw a rectangular box with two opposing corners at the given points (x_1, y_1, z_1) , (x_2, y_2, z_2) .

Table D-7. Boolean region methods provided by all geometric primitives and the Region class

Method	Usage	Description
Union	<code>a.Union(b)</code>	Region union: the set of all points contained in either a or b.
Intersect	<code>a.Intersect(b)</code>	Region intersection: the set of all points contained in both a and b.
Minus	<code>a.Minus(b)</code>	Region difference: the set of all points in a that are not contained in b. Equivalent to <code>a.Intersect(b.Not())</code> .
Not	<code>a.Not()</code>	Region complement: the set of all points not contained in a.

Table D-8. Methods provided by the Object class

Method	Arguments	Description
push	(region, material ID)	Set the given region of space to contain the given material ID.
set_bbox	$(x_1, y_1, z_1, x_2, y_2, z_2)$	Set the object bounding box to have corners at the given points (x_1, y_1, z_1) , (x_2, y_2, z_2) .
set_dims	(n_x, n_y, n_z)	Set the x, y, z extents of the object discretization.

<code>write_obj (filename)</code>	Write the material ID data to the given file.
<code>write_vol (filename)</code>	Write the volume fraction data to the given file.
