Computer Society International Design Competition 2006
Final Report

Team Members

Hunter Davis
hkdavis@ncsu.edu
Computer Science

Josiah Gore
jjgore@ncsu.edu
Computer Science

Eric Helms
edhelms@ncsu.edu
Physics

Blake Lucas
bclucas@ncsu.edu
Computer Science
Electrical & Computer Engineering

Abstract

The scientific and medical communities are well aware that ultraviolet radiation can be detrimental to human health; however, it is unknown how much ultraviolet radiation people receive from sun exposure and what dosage is required before the sun becomes hazardous to human health. Current methods for determining the amount of solar radiation on the human form are crude and unrealistic approximations. So far, the best approach has been to place polysulfone patches, which react to ultraviolet radiation, on mannequins and expose the mannequins to the sun for several days. These studies required time-consuming preparation and analysis and were poor approximations of typical sun exposure. Scientists who study the effects of sun exposure have asked for a better tool for measuring the amount of radiation received by a 3D form under given temporal, spatial and atmospheric conditions. The SunRay system couples ray-tracing with a numerical solar radiation model to calculate the amount of radiation that falls on a 3D form. The system is intended to be used as a research tool to calculate, analyze and visualize solar radiation data for specified scenarios. The system also pulls atmospheric and geographic information from a worldwide network of pyranometers and weather stations to use as input. SunRay uses a validated, open source radiative transfer model to calculate solar radiation across several specific wavelengths that affect human health. The SunRay application provides an adjustable scenario control interface that gives the user the flexibility to run a range of tests and visually inspect the resulting data.

System Overview

Background. Solar energy sustains life on Earth. The visible region of the solar spectrum drives photosynthesis in plants. Organisms higher in the food chain ultimately derive their energy from plants. The extraterrestrial spectrum of the sun's radiation is, however, incompatible with life. Solar radiation known as UVB is partially absorbed by stratospheric ozone. Pre-industrial levels of stratospheric ozone sufficiently diminished surface irradiance of harmful UVB, allowing plant and animal life to flourish. Since the 1930's ozone depletion from chlorofluorocarbons (CFCs) has resulted in an increase of UV energy that strikes the Earth's surface [1]. The World Health Organization recognizes UV radiation as a human carcinogen [2]. Today, skin cancer affects more than 1 million Americans each year and accounts for 10,000 deaths annually, which is approximately 4% of all cancer deaths [3]. Additionally, solar UV radiation contributes to cataracts, which are the leading cause of blindness worldwide [4]. The 1.2 million cataract surgeries performed annually are Medicare's largest line item expenditure [3]. The exact percentage of cataracts that are caused by UV radiation is unknown; however, it is clear that increased UV exposure has significant implications for virtually all terrestrial ecosystems and the life that inhabits them [5].

While the effects of solar UV radiation on living tissue are well documented, estimates for the total actual dosage received by people, and by plants and animals, have been severely limited. These estimates are limited due to the complex 3D geometry of human, animal and plant forms and mobility of humans and animals. Researchers have attempted to measure sun exposure using polysulfone radiation detectors placed on mannequins. These mannequin studies provide the majority of what is known today about human exposure to solar UV and have demonstrated non-uniformity in sun exposure across the human form [6]. Although these time-consuming studies were thorough, they were necessarily limited to specific postures, orientations, forms and atmospheric conditions. The mannequin studies did not address how radiation dosages change when a person engages in different activities at different locations at different times of year. Moreover, this measurement approach cannot accurately predict future sun exposure levels caused by further stratospheric ozone depletion. An alternative approach to measuring radiation exposure was published recently [7]. In that paper, the authors describe an algorithm to estimate solar radiation on a 3D human form using a numerical approach to solar radiation exposure simulation modeling. The SunRay system presented here builds upon this numerical simulation approach. Our goal is to provide a research tool for scientists to study sun exposure on living organisms. We believe this physically-based research tool will contribute to a more accurate assessment of sun exposure so that scientists can make the best case for societal changes that preserve, protect and enhance the environment.

The SunRay System. The SunRay system estimates radiation exposure across multiple spectral bands for a given 3D form under specific temporal, spatial and atmospheric conditions. The system collects atmospheric data from instrumentation to provide real-time input for the simulation. For each time step in the simulation, a ray-tracer determines which sun rays can hit the 3D model. Based on inputs specified by the user, the Solar Radiation Model (SRM) calculates the energy delivered by each sun ray to determine the dosage of solar radiation received on the 3D model. The output of the simulation is a graphic visualization of sun exposure levels on the 3D model and a corresponding data file.

Innovation & Benefits. The SunRay system enhances and extends existing sun exposure measurement approaches by incorporating real-time data, a diffuse radiation model, multiple

spectral bands, and extensibility to a range of research investigations. SunRay is superior to empirical studies in its ability to easily manipulate scenarios, to collect results quickly, to predict radiation exposure for future ozone levels and to model both the direct and diffuse components of solar radiation at discreet spectral bands. The SunRay system can be used to address several questions of interest to the scientific community and the public. Scientists are interested in accurate measurements for the Total Body Burden (TBB) of radiation for humans and other species, which is correlated with systemic immune system suppression. SunRay could be extended to measure sun exposure for specific regions of the body, such as the eyes, head, chest and feet. This information can be used to communicate to the public the dangers of sun exposure and raise awareness of the discrepancy between actual UV exposure and the perception of UV exposure inferred from sunlight brightness and radiant heat. Finally, the modular and hierarchical design of the SunRay system permits it to be extended and upgraded to include more complex scenarios and additional phenomena.

Performance & System Requirements. In order for SunRay to be a valuable research tool, the system must meet several performance requirements important to the scientific community. SunRay must have an intuitive interface that allows the user to specify settings for the simulation scenario. In addition to a Graphical User Interface (GUI), the system must read settings from hardware devices and execute batch runs of specific scenarios. SunRay software should be optimized for speed so that researchers can simulate many scenarios in a reasonable amount of time. SunRay should utilize an efficient ray-trace algorithm and attempt to minimize the number of ray-traces required for a particular scenario. SunRay should also provide visual and numerical results so that the results can be easily inspected and accurately analyzed.

Design Methodology. The SunRay system was developed using an iterative software development methodology. This methodology was chosen so that the SunRay system would evolve gradually as new functionality was added and tested during each iteration. Before development began, the team collaborated on defining system requirements and compartmentalizing the design of the system into independent modules with well defined interfaces. Over the course of the next 14 weeks, four iterations led to a final product. At the beginning of each iteration, we identified and documented target system requirements for the iteration. A system design was developed and presented via UML diagrams. The entire team collaborated on the design of high-level system and module interfaces. However, internal design of modules was left to individual developers to implement their assigned requirements and ensure that there was traceability from their requirements to design, implementation and testing.

The SunRay system was built in layers utilizing existing open source software. During the first iteration, base-line functionality was developed for visualization, 3D model manipulation and a GUI. In the second iteration, ray-tracing, SRM interfacing and integration of the visualization engine with the GUI and ray-tracer were implemented. During the third iteration, the SRM was integrated with the visualization engine and ray-tracer. During the final iteration, the system was completely integrated with the sensor interface and subjected to tests to verify that all requirements were met.

The development team was composed of four members. Hunter Davis, a computer science major, is responsible for GUI development. Josiah Gore, a computer science major, is responsible for generating output data and its visualization. Blake Lucas, a computer science, electrical engineering and computer engineering major, is responsible for the ray-tracing module. Eric Helms, a physics major, is responsible for the radiative transfer model.

Implementation and Engineering Considerations

SunRay's system requirements were developed after consultation with the scientific community regarding how best to meet the project objectives. We believe these requirements are sufficient for SunRay to become a viable research tool for estimating sun exposure on 3D forms.

System Requirements

1. The system shall have an interface that allows user inputs for the time, date, ozone level, aerosol optical depth (AOD), 3D form, orientation and location, weighting function and diffuse grid resolution.
2. The system shall have an interface for communicating with hardware sensors, particularly a pyranometer.
3. The system shall be optimized such that many simulation scenarios can be run in a reasonable amount of time.
4. The system shall calculate total irradiance for multiple distinct radiation bands.
5. The user shall be able to adjust the spatial and temporal granularity of the simulation.
6. The system shall use a ray-tracing algorithm that minimizes the number of ray-traces to determine the distribution of diffuse and beam radiation on a 3D model.
7. The system shall allow the user to manipulate the camera position for the 3D model output interactively.
8. The system shall provide 3D visualization and data inspection of irradiance results.
9. The system shall provide XML files of the radiation intensity for one or more spectral bands.

Overall Design

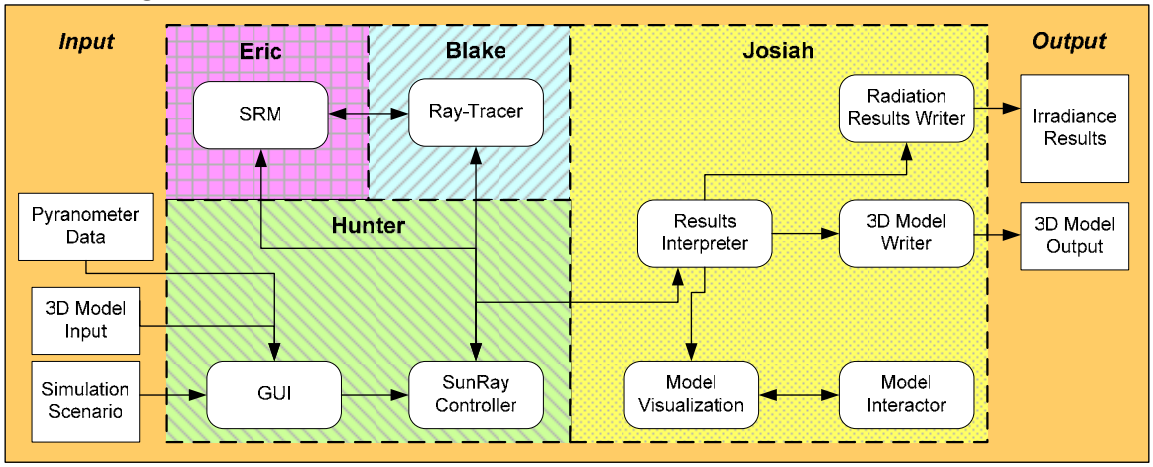

Figure 1: SunRay system flow diagram.

Figure 1 illustrates the overall system flow and the division of responsibilities between the four developers. First, the system reads pyranometer data from the interface. Next, the user selects scenario settings and a 3D model file for the simulation and selects to start the simulation. The GUI sends input parameters to the SunRay controller, which initializes the SRM and ray-tracer. The Ray-tracer then traces the 3D model and calculates the irradiance based on values from SRM. The results are returned to the SunRay controller, which proceeds to increment the time step and repeat the ray-trace process. At the end of the simulation, the results are sent to an

interpreter that writes the irradiance values to a file and converts the values to colors. Finally, the 3D model is shaded with the color values, displayed using the visualization module and written to a 3D model file.

Sensor Hardware

A pyranometer is a sensor that measures irradiance on a horizontal surface (see Figure 2). The simplest pyranometers measure broadband irradiance while more sophisticated pyranometers can measure spectrally resolved irradiance. Normal incidence flux can be calculated via broadband measurements while total column ozone and AOD can be derived using spectral band ratio algorithms.

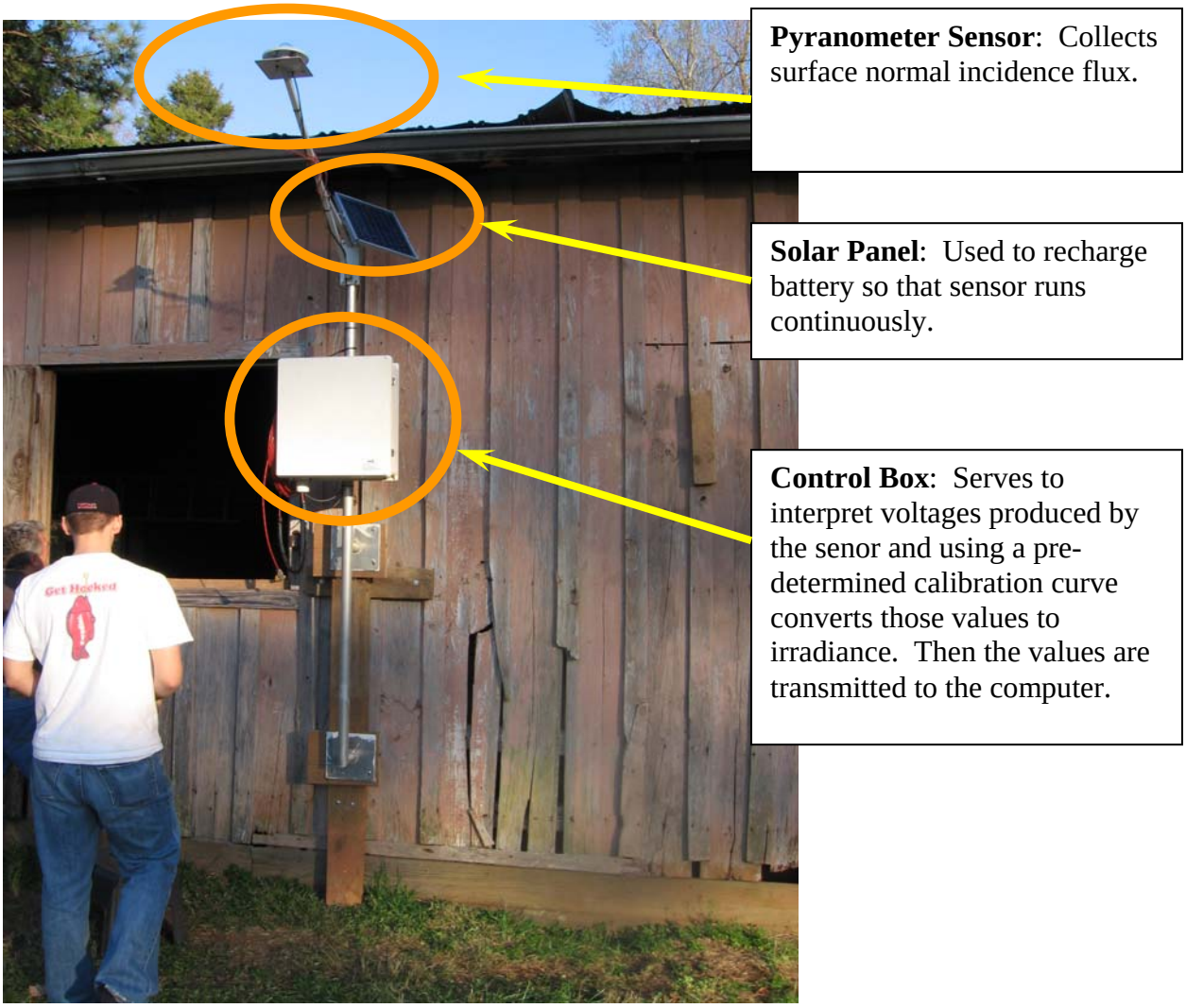

Figure 2: The pyranometer apparatus with the sensor, solar panel and control box.

As a first step, a simple pyranometer(referred to below as the NCSU pyranometer) was installed by our team. A homemade base was fabricated and attached to the side of a building in order to structurally support the weight of the pyranometer and place the pyranometer as high as was possible (see Figure 3). The location of the pyranometer was chosen to maximize the amount of visible sunlight during peak hours of the day. As seen in Figure 2, the sensor is

mounted at the apex of the assembly with the solar panel below, but high enough to get enough sunlight within a day in order to keep the battery charged. The white control box contains a 12-volt battery along with a Sutron 9210 Data Logger. A SolRad sensor block is used to collect input from the pyranometer and transfer it to a computer. Data is collected at a rate of 1 sample/sec, and a nearby computer logs this data in a XML file by polling the pyranometer. The log can then be read off manually and parsed to extract the broadband horizontal surface flux for a given time of day. Dividing this value by the cosine of the solar zenith angle gives the normal incidence flux, which represents the maximum value that any point on the body can receive.

Figure 3: The team custom building a base for the pyranometer so that it gets maximum sun during the peak hours of the day.

After experimentation with the NCSU pyranometer, we discovered that more sophisticated pyranometers of a USDA network provided real-time data values for the total column ozone and AOD. The USDA operates the pyranometer network via a website and gives downloadable data from the previous day [3]. While the primary objective of the USDA network is to monitor UV and total column ozone for agricultural applications, our SunRay system is extending the use of this data to a human health application. The values of total column ozone and AOD, combined with the geographic location of the pyranometer in the world can be used as real inputs to the SunRay system to create accurate estimates.

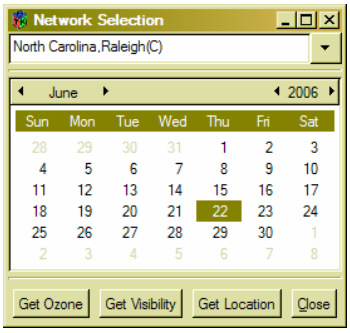

Figure 4: Network Interface GUI

Figure 4 depicts the GUI interface used to pull values from the USDA website without using a web browser. By creating an interface to automate the USDA website, data from one or many locations (e.g., Regina, Saskatchewan, Davis, CA, Fairbanks, AK, and Alexandra, New Zealand) can be used as input into the scenario. This input data includes the longitude and latitude of the sensor location, time of day, month and year for which the data was taken. The real-time AOD (i.e. visibility) and total column ozone are input to the regression equations.

Software Design

Software Architecture

SunRay's software was designed to maximize the amount of code that could be developed independently among four team members. The system architecture consists of five tiers: Visualization, Network, GUI, SRM and Ray-tracer (see Figure 5). Modules in each tier were built upon existing Open Source Software (OSS) and developed independently of the other tiers. In order to complete this project in three months, it was crucial that we utilize OSS to accelerate development time. In order for SunRay to serve as a research tool, it was also essential that we use a modular design to allow the system to be modified and extended easily. For instance, the SRM can be updated by members of the scientific community without necessitating changes to other modules in the system. The layered architecture allows future developers to scale the SunRay system to more complex simulations that attempt to model typical activities of individuals who are exposed to the sun for prolonged periods of time.

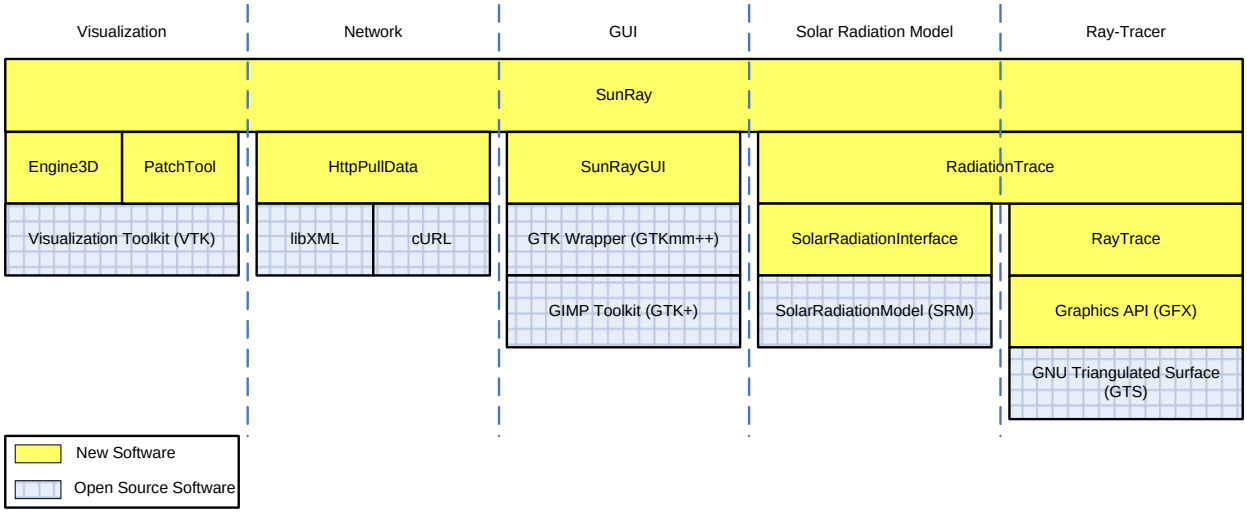

Figure 5: Software architecture of SunRay.

Sensor Interface

In order for SunRay to use the data produced by the pyranometer an interface had to be created. The scenario input can either come from user selected input or a pyranometer selected from the network. To interact with the pyranometer network, an http interface was written to access the real-time data across the internet.

Graphical User Interface

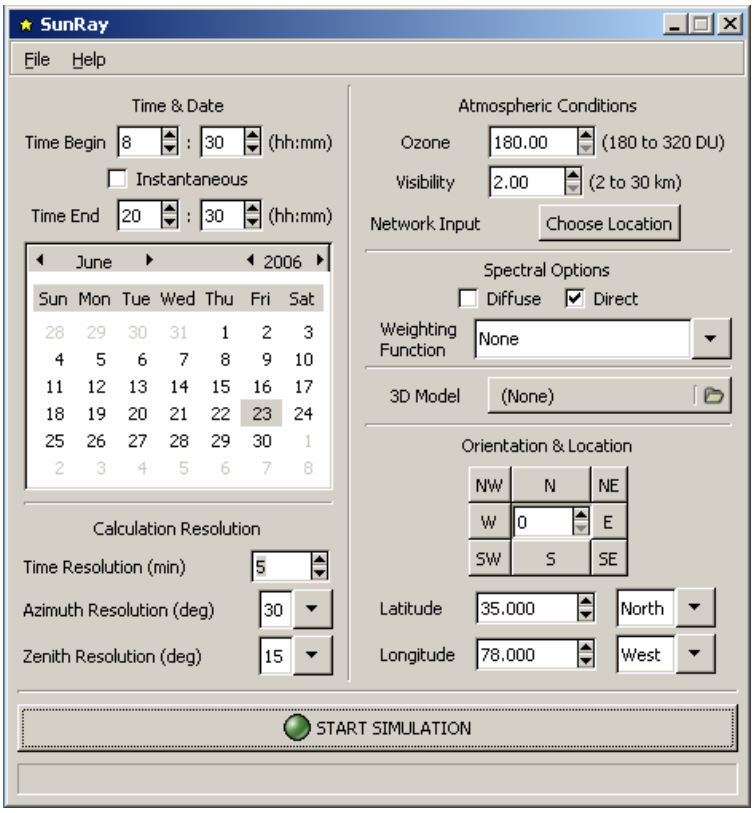

Figure 6: The GUI

The GUI was developed in consultation with the scientific community who will ultimately use our tool. The start time and date of the simulation are required inputs, and the user can choose whether to run an instantaneous or finite simulation. If finite, the end time and time resolution are required. Other necessary inputs include diffuse grid resolution, total column ozone, AOD, the 3D mesh file, orientation and location, weighting function, and spectral options (see Figure 6).

The time resolution is the length of the intervals that the total simulation duration is broken into. A resolution of five minutes means that the sun's position will be calculated and an irradiance value found at each five minute interval. Because the total duration of the simulation must be divided into equal sections, the time resolution must be a factor of the number of minutes in the total simulation. An error is displayed to the user if this condition is not met. The diffuse grid resolution represents the number of degrees that the visible sky is broken into for the azimuth and zenith directions. In order to compromise between data precision and speed of the calculations, the diffuse grid and time resolutions are presented to the user. The user can

decrease the angular resolution of the diffuse ray-trace to reduce computational time. Alternatively, the accuracy of the diffuse ray-trace can be improved by increasing angular resolution. These tradeoffs were provided so that users can get a quick qualitative idea of the solar radiation dosage being received.

For the 3D mesh, the users can also select which mesh file to use, geographic location (latitude and longitude), and orientation, which is the geographical direction the model faces. The parameters for the SRM include selecting a spectral weighting function, the AOD, and total column ozone levels. The user is able to choose from a range of 180 to 320 Dobson units for the concentration of column ozone. The AOD is adjustable between 2 kilometers and 30 kilometers. The system has the ability to calculate diffuse irradiance, direct irradiance or both, and the users can specify which they want. If neither is chosen, the system will alert the user with an error message.

To interface with the network pyranometer, the user clicks a button in the GUI that opens a secondary window (see Figure 4). This window provides a list of available network sites, as well as another calendar allowing the user to change the date without switching windows. The user is provided with three options, obtaining total column ozone, AOD, or the location of the pyranometer. By selecting one of these buttons, the application will download the appropriate values from the network, if available, and populate the GUI with these values. An error message is displayed to the user if the chosen value is unavailable.

Numerical Simulation

GFX. The GFX Application Programming Interface (API) was a tool developed specifically for SunRay to provide the functionality necessary to implement the ray-trace algorithm we selected. GFX was built on an existing graphics API called the GNU Triangulated Surface (GTS) library. GTS provides all the functionality necessary for the ray-tracer, but because the library was written in C, it was difficult to use with our SunRay application, which is written in C++. GFX alleviated implementation difficulties by wrapping each C structure with a C++ class so that 3D models could be manipulated using object-oriented code. However, the wrapping processes turned out to be difficult process because C structures had to be created, initialized and destroyed in sync with the C++ classes. Moreover, multiple objects stored references to the same object, so it was not always appropriate to destroy all member objects associated with each class. For instance when a triangle is destroyed, not all the vertexes can be destroyed because multiple triangles can have a common vertex in a 3D model. Although GFX was a challenge to write, it ultimately saved us time when developing the ray-tracer because we could apply object-oriented concepts, such as inheritance, abstraction and encapsulation, to accelerate development.

Ray-Tracer. For direct and diffuse irradiance calculation, SunRay utilizes the ray-trace algorithm described by Matt Pharr and Greg Humphreys in their open source ray-tracing package PBRT [8]. To meet performance requirements for SunRay, we chose PBRT because its ray-tracing algorithm is optimized for complex models that have multiple light sources, which is the case for our SunRay application. However, PBRT was developed as a plug-in for Maya 3D rendering software and not intended for standalone use. Since SunRay required only a small portion of PBRT's functionality, we decided to rewrite the core ray-tracing algorithm using our own graphics API instead of modifying the PBRT package for this application.

The efficiency of all ray-tracing algorithms hinges on the ability to minimize the number of intersection tests required to determine if and where a ray intersects a 3D model's surface.

PBRT utilizes a K dimensional Tree (Kd-Tree) of triangular faces derived from the 3D model to minimize the total computational cost of intersection tests for each ray cast.

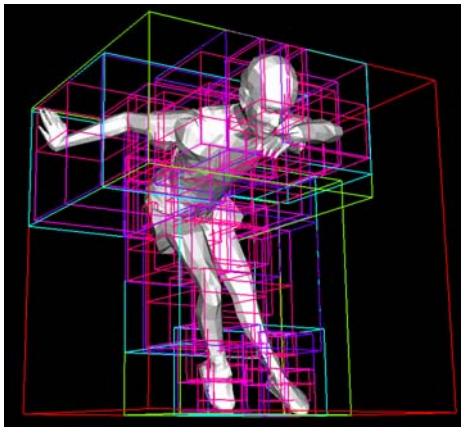

Figure 7: Bounding Box example.

A Kd-Tree is simply a tree where each node represents an Axis-Aligned Bounding Box (AABB) that encloses a set of triangular faces (see Figure 7). The root node encloses all triangular faces. Each child node encloses a subset of the parent node's faces. The leaf nodes of the Kd-Tree each contain a single face.

Ray-AABB intersection tests are significantly less computationally intensive than ray-triangle intersection tests. Therefore, the Kd-Tree attempts to minimize the number of ray-triangle intersection tests by performing cheaper ray-AABB intersection tests for interior nodes and ray-triangle intersection tests for leaf nodes. During top-down Kd-Tree construction, the objective is to split a node perpendicular to the X, Y or Z axis to create two child nodes whose combined estimated intersection cost is less than the intersection cost of the parent node. PBRT uses the following metric that incorporates the cost of traversing the tree (*traversalCost*), the cost of a ray-triangle intersection (*intersectCost*), the number of faces in each child node (N_A, N_B) and an estimate for the probability that the ray will intersect either the left or right child (p_A, p_B) (see Equation 1).

$$S_A = Surface\ Area\ Of\ Child\ A$$
$$S_B = Surface\ Area\ Of\ Child\ B$$
$$S_P = Surface\ Area\ Of\ Parent$$
$$p_A = \frac{S_A}{S_P} \qquad p_B = \frac{S_B}{S_P}$$
$$Cost(P) = intersectCost \cdot N_P$$
$$Cost(A,B) = traversalCost + intersectCost \cdot (p_B N_B + p_A N_A)$$

Equation 1: PBRT ray-tracing algorithm.

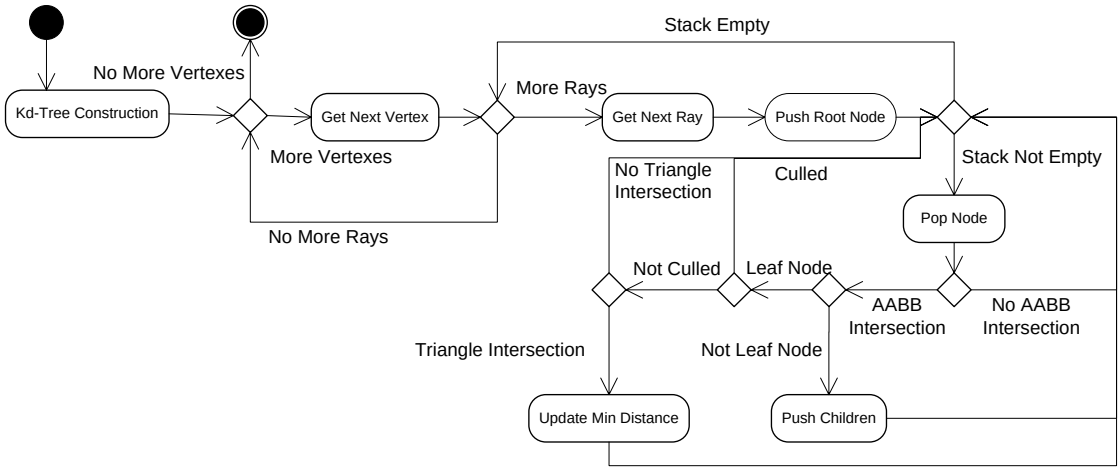

Figure 8: Ray-tracer state chart diagram.

The ray-tracing algorithm is depicted in Figure 8. For each ray-trace, the Kd-Tree is traversed in a depth-first order starting at the root node, and a ray-AABB intersection test is performed for each node. If the ray intersects a node's bounding box, then the node's children are traversed. If the node is a leaf node, then a ray-triangle intersection test is performed for all faces enclosed by the node's bounding box. If the ray does not intersect any of the faces in a node, then the search continues. If the ray does intersect a triangle face inside a leaf node, then the minimum distance along the ray to the surface of the object is updated and the search continues for a shorter distance along the ray to the surface. Since nodal bounding boxes can overlap, faces are tagged after an intersection test so they are not tested again for the same ray. In addition, faces are cull tested to check if the ray could intersect the front side of the face. The cull test can reduce the amount of ray-triangle intersection tests by half for closed 3D models, which is the case for most models used by SunRay.

Solar Radiation Model (SRM). The spectral solar radiation model (SRM) uses the radiative transfer algorithm from the LibRadtran model [9]. LibRadtran calculates diffuse and direct beam irradiance from 100nm to 800nm in 1nm bandwidths. In addition to spectral calculations, SRM accounts for total column ozone and AOD in the regression equations for radiation. SRM also provides the capability to calculate biologically-effective irradiance and exposure by convolving spectral weighting functions with the irradiances/exposures at each wavelength. The model currently has weighting functions for the following outcomes: erythema, DNA, skin cancer, melanoma, cataract, immuno-suppression, plant growth and photoplankton. An exposure model with spectral resolution is important because photobiological reactions, such as skin cancers, cataracts, immunosuppression, or foliar necrosis, are wavelength dependent. Since most of the SRM model was written in FORTRAN, a C++ interface was written to interface with the FORTRAN subroutines.

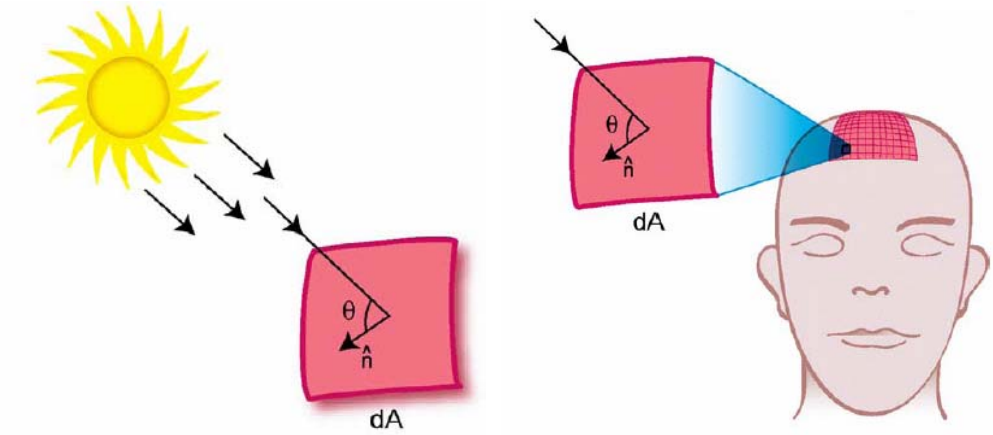

Figure 9: Depiction of a sun ray hitting the top of a person's head [7].

Irradiance is power density ouput by the sun incident on a surface. Direct irradiance is found by computing the dot product between the irradiance vector for the direct sun ray and the normal to the surface of the polygon (see Figure 9). Diffuse irradiance is found by integrating the radiance contained in each diffuse sun ray over the hemispherical field of view for each polygon. Radiance, the brightness at a given point in the sky, is dependent on the angle between a particular sun ray and the location of the sun. Direct and diffuse irradiance can be summed to compute the total irradiance.

System Output

Once the results have been computed for a mesh, SunRay presents a 3D interactive display window that gives the user control over the view. Using only the mouse, the camera can be rotated and zoomed, giving the user the ability to examine the mesh from all angles, as well as direct the camera onto a specific area of the mesh. This ability to focus on and enlarge a certain area is made extremely quick and simple with a "fly to" feature, through which one keypress moves and turns the camera toward the geometry pointed to by the mouse. Another interactive feature provided to the user switching between wireframe and full surface rendering. A full visualization of spectral solar radiation data requires a more sophisticated approach than the strict use of false color mapping. Since it would be difficult to simultaneously visualize all wavelengths on the 3D model, the display will allow the user to select one or multiple wavelengths to visualize the effects of different spectral ranges. An additional feature of the display interface is the display of directional axes to provide a sense of orientation to the mesh. XML

Software Integration

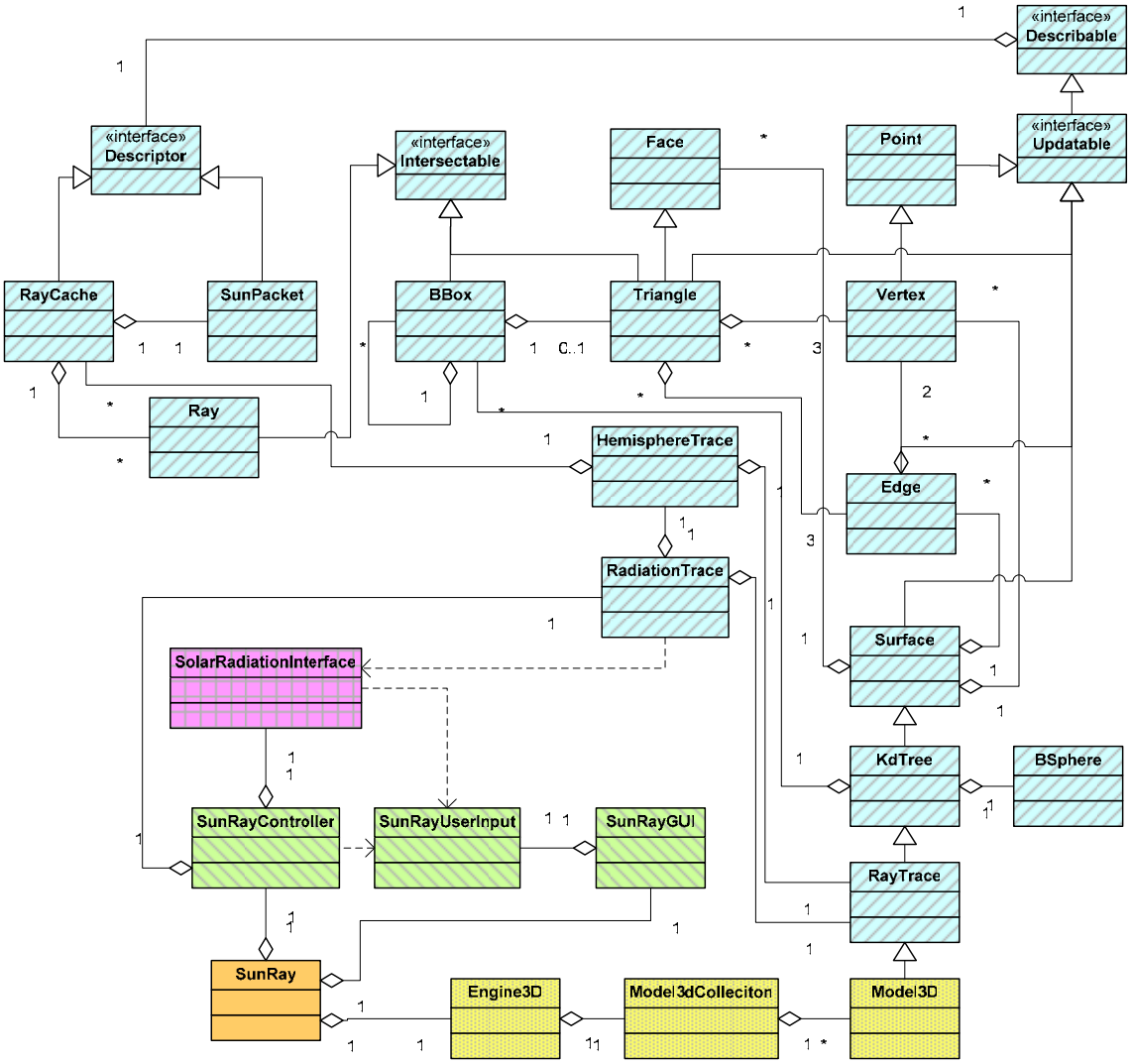

Figure 10: Primary Classes of SunRay.

Figure 10 shows the class level integration of the four primary SunRay components: GUI, SRM, Ray-tracer, and Visualization. The SunRay execution process begins by creating an instance of a GTK window, which allows the user to input values to define the simulation. When the user presses the start button, the parameters are passed to the Solar Radiation Interface, and control passes back to the SunRay main controller. The RayTrace class builds a KdTree on the 3DModel specified by the user in the GUI and creates an instance of the RadiationTrace class. RadiationTrace accesses the data provided by the SolarRadiationInterface to calculate flux values while doing the ray-tracing calculations. The final results are sent to Engine3D for visualization. Utility modules, not detailed in Figure 10, were created to provide enhanced visualization tools and a USDA network interface.

Verification and Testing

The user interface and visualization modules were tested through extensive and repeated user interaction. Boundary values were used to test the constraints placed on the GUI components. During this phase, inputs on were printed to standard output to make certain they

are true to those the user specified. The visualization, which involves mouse and keyboard shortcuts, was given every possible keypress and mouse click to determine any flaws in the interaction. Initially, panning was included in the camera interaction, but testing indicated that this functionality could easily allow the user to lose track of the model while not providing appreciable benefits. As a result, this feature was removed from the system.

Having tested these individual pieces for correct functionality, integration became our next priority. The first working full builds were run through a wide variety of scenarios, and the magnitude of the results was monitored in order to verify that they were reasonable. For our purposes, we defined reasonable magnitudes as being roughly between 0 and 9000. Other verification of the data included ensuring that for a given vertex, spectral data followed expected patterns, with larger values occurring in the visible wavelengths than outside it, for example.

Aside from ensuring that correct results were given in typical scenarios, we tested the bounds of the system by providing it with extreme values. Some of these tests were simple to verify. If the user selects a time in the early morning before the sun has risen, the flux values should obviously be negligible. The effects of extreme values for other variables were more difficult to verify. With assistance of an expert from the scientific community, we made certain that we only allowed the user to select reasonable bounds for the atmospheric conditions, and that using values at the boundaries resulted in realistic output.

At the beginning of the GUI development phase, we consulted with the scientist to determine which inputs were necessary. At several vital stages during implementation, we consulted with the sample user again to make sure the layout and input values were intuitive and useful to the user. Some changes suggested by the user were to give an easy option for setting model orientation to one of the eight primary directions, and to make the selection for beam, diffuse or both simpler.

To test the GUI, we used command line output with C++ console output commands. We selected values from the GUI and compared parsed results to the selected values. At each stage of testing we used the same procedure to ensure that the correct values were being used. Boundary testing of input values was simplified by Glade's WYSIWYG editor. For instance, GTK spin buttons can be initialized with upper and lower boundaries, and combo boxes with a list of options. This kept the amount of code needed to bound input values very low.

The ray-tracer was tested via visual inspection of the Stanford Bunny [10] and seashell [11] models after false-color rendering (see Figure 11). The 3D models were specifically chosen because the expected results could be easily predicted, and the models are sufficiently complex to demonstrate the robustness of the ray-tracer. To test direct beam visibility, vertexes intersected by sun rays were colored red, such as the bunny's back, and all other vertexes were colored green, such as the inner bunny ear. When the model is viewed from the sun's direction, the model appears all red and appears green when viewed from the opposite direction. To test diffuse functionality, the number of ray intersections were counted for each vertex and mapped to different colors. For the seashell model, the interior of the shell receives few sun rays, indicated by orange and red, while the exterior of the shell receives many sun rays, indicated by yellow and green.

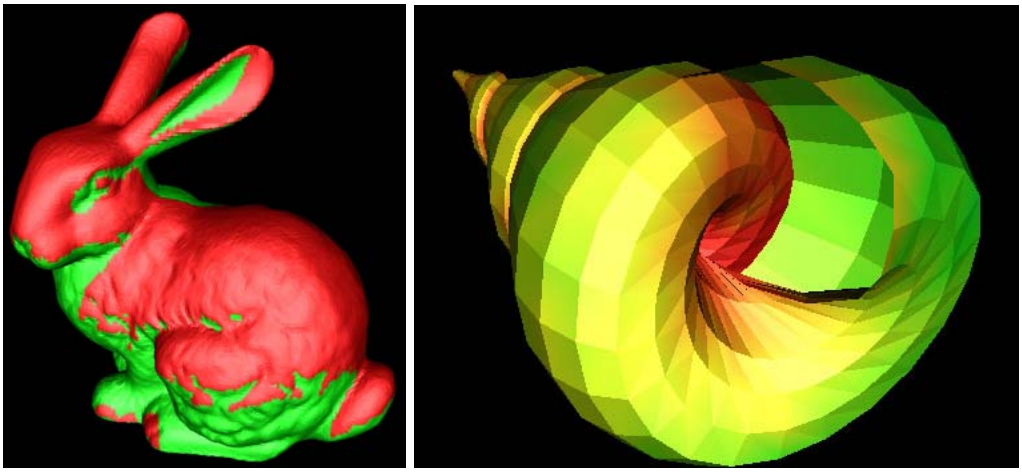

Figure 11: Ray-Tracer Testing.

Results

Data results from the simulation are rendered as a false colored 3D form and written to a XML file. The false color rendering below shows how the irradiance values vary across the entire form of the body (see Figure 12). The scaling ranges from blue being the minimum irradiance and red being the maximum. Below The 3D model, there are histograms detailing the distribution of power density across all spectral bands. There is also a plot of the average power density per band and a table of irradiance values.

The spectral band and patch selection GUI allow the user to inspect the data (see Figure 13). The spectral band selection component allows the user to display certain spectral bands on the model and on the graphs (see Figure 14). This functionality improves readability and allows the user to focus on the spectral range of interest. The patch selection component allows the user to add, delete or select a patch. A patch is a subset of triangle faces on the 3D model. If a researcher is only interested in a certain region of the model, the patch selection component will allow the user to select that region. The irradiance is then recalculated for only those selected regions and displayed in the tables and graphs surrounding the 3D model visualization (see Figure 15).

The XML files represent the diffuse and/or direct beam irradiance calculations and the data provided via the NCSU pyranometer. These XML files allow for easy importation into a spreadsheet program such as Microsoft Excel which allows for production of graphs of the entire range of values for every vertex or all wavelengths for a particular vertex by selecting a single row.

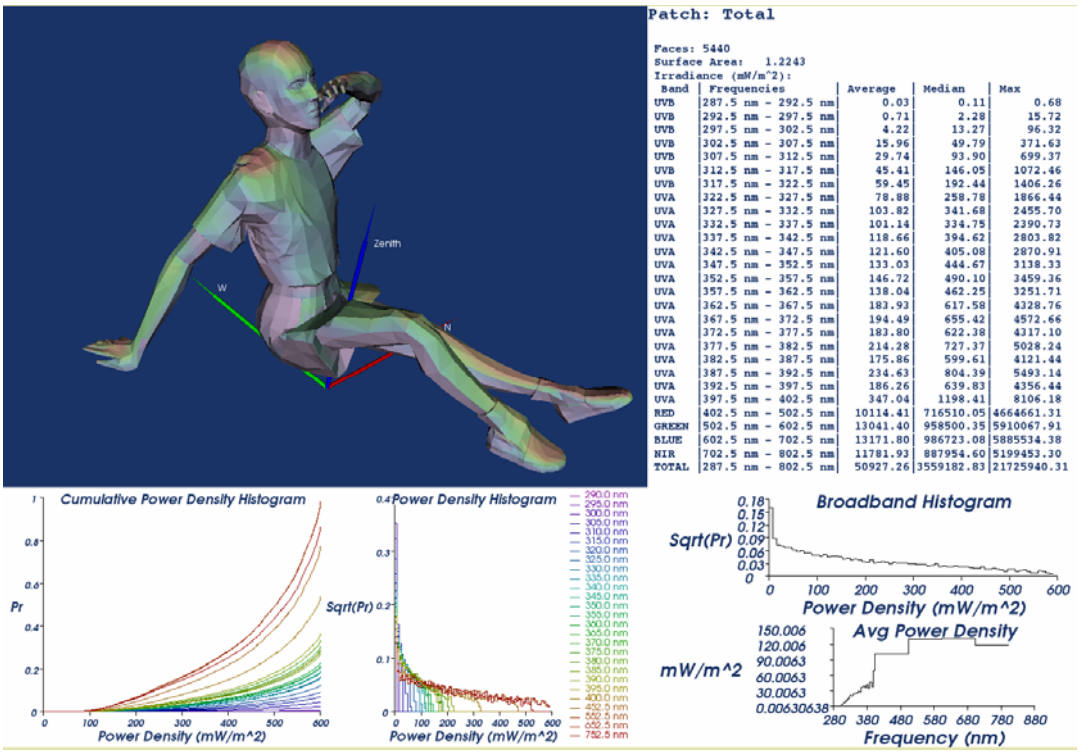

Figure 12: Visualization of Total Body Irradiance

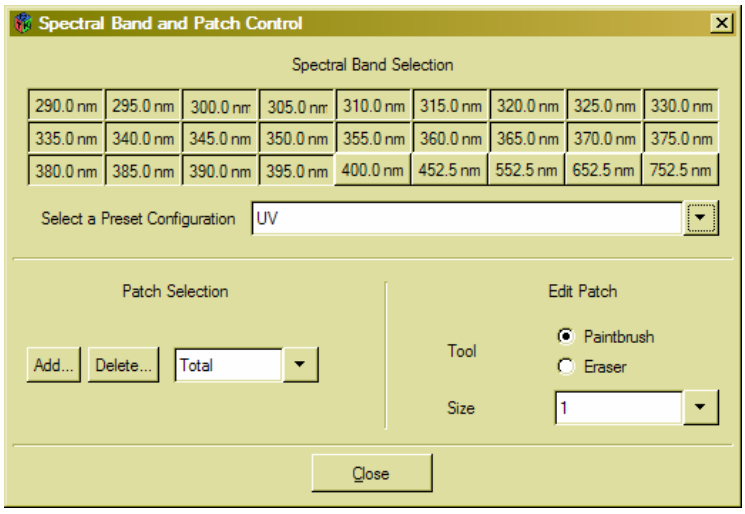

Figure 13: Spectral Band and Patch Selection GUI

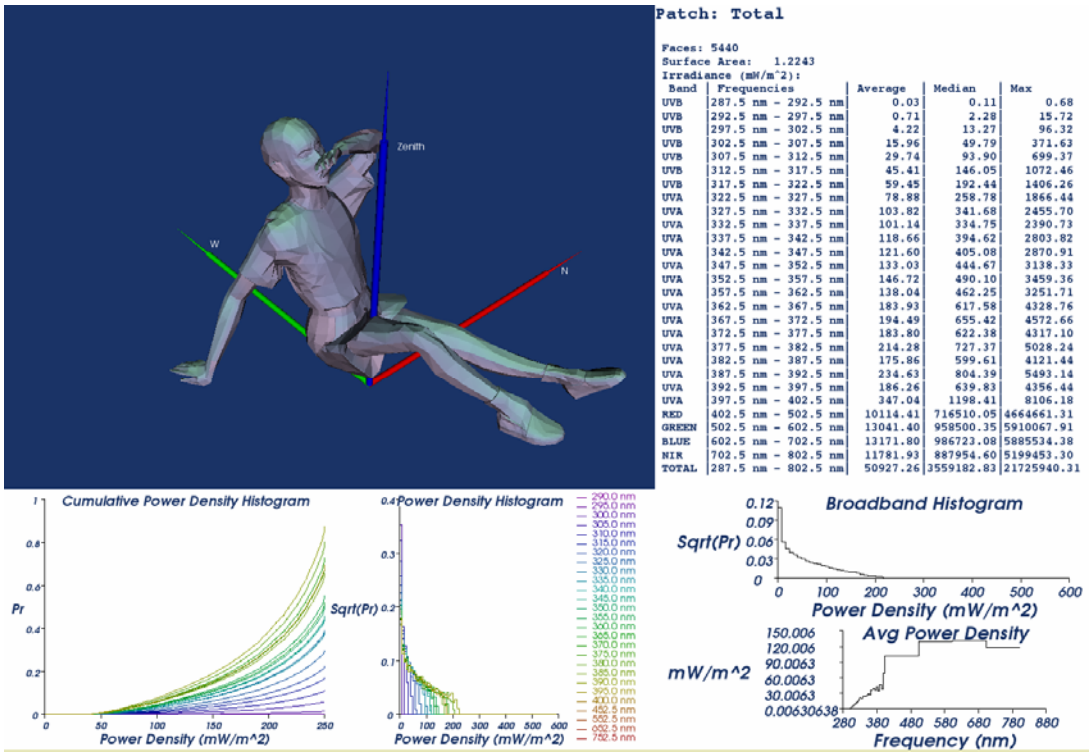

Figure 14: Example of UV Band Selection

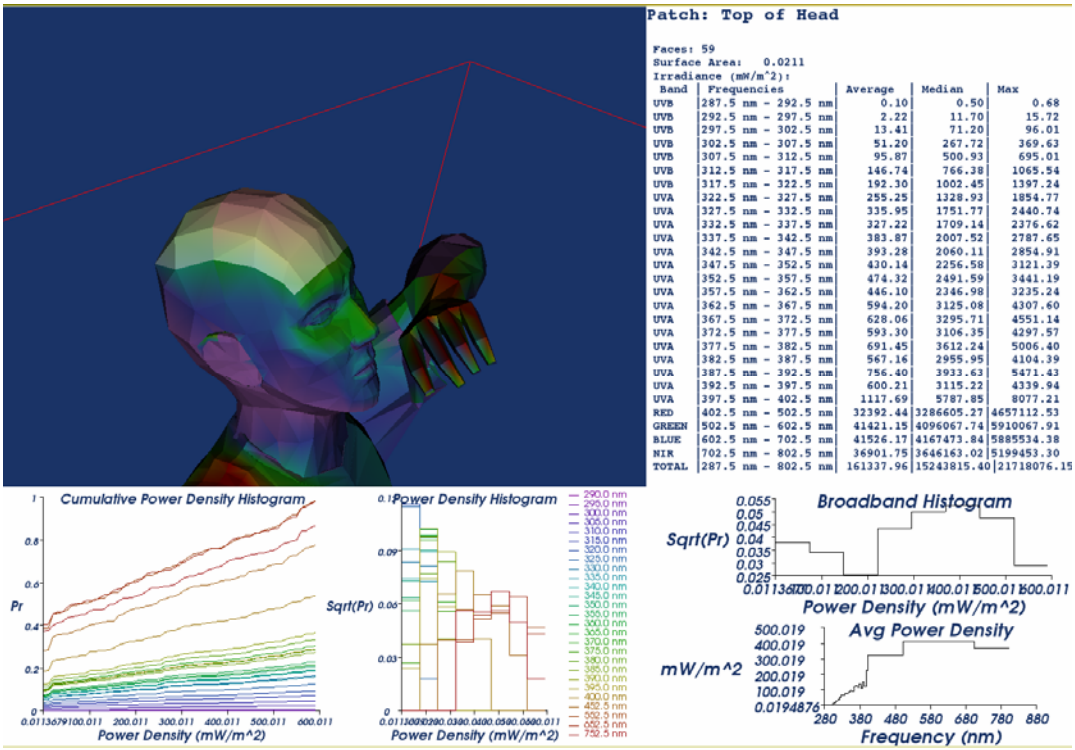

Figure 15: Top of Head Patch Selection

Project Management

The SunRay project consisted of four milestones that were scheduled to be completed at the end of each sprint, except for the last milestone which marked final testing and demonstration of the system. The project schedule included extra time in case a task took longer to accomplish than predicted. Developers often had to work longer than the scheduled 30 hours per sprint in order to implement all their requirements for each milestone. All milestones were met a few days before the scheduled completion of each sprint (see Figure 16).

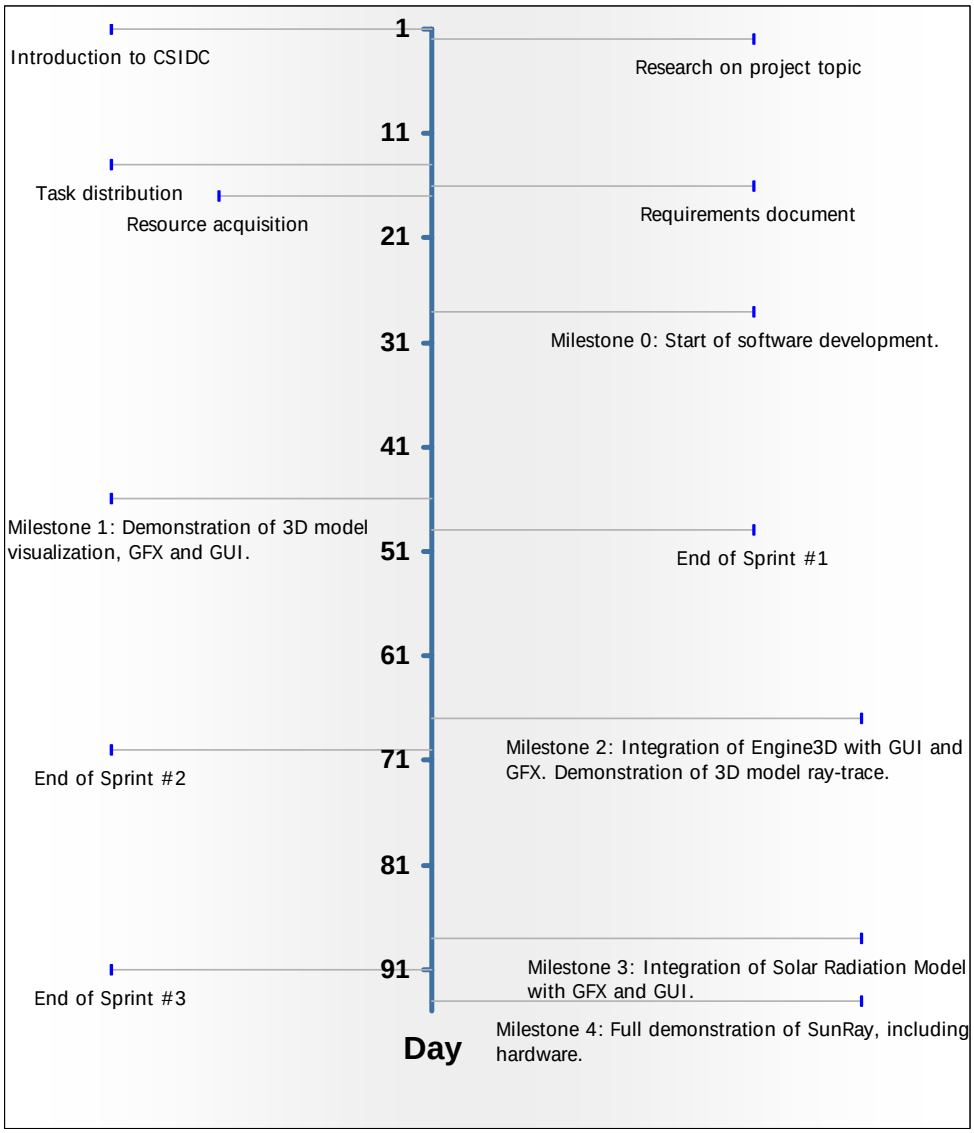

Figure 16: SunRay project timeline.

Scrum Technique

The Scrum technique was used to manage the iterative development process effectively [12]. Scrum's key practices emphasize individual interactions, working software, customer collaboration, and responsiveness to change. The product backlog is at the core of Scrum management. Once system requirements were identified, they were prioritized and documented in a product backlog. The project proceeded in twenty-one day sprints. The highest priority features were removed from the backlog, implemented and demonstrated at the end of each

sprint. Milestones encompassed the amount of work that was predicted to be accomplished by the end of each sprint. Each developer was expected to implement 30 hours of work in the backlog per sprint. At least once per week, the team met to discuss the work they accomplished since the last Scrum meeting, problems they encountered, and what they will do for the next Scrum meeting. Daily progress on the project was documented in written logs by each developer.

Tradeoffs

When installing the local pyranometer, there were logistical decisions to be made concerning the location, position and networking of the installation. It is necessary that the sensor not be shaded during daylight hours, which meant it would have to be mounted high up above nearby obstructions such as trees and buildings. Before development of our system began, the sensor was mounted on top of a building on NCSU campus. This location was adequate, but it did not allow for access by team members, as only maintenance personnel can access a roof.

We chose a location in North Raleigh, NC, because it was isolated enough to provide an unobstructed view of the sky, protected against tampering and vandalism, and was accessible to the team. We had to build a structure to support the weight of the sensor and raise it high enough to prevent the tree line from blocking sunlight. A nearby building provided shelter for the PC that accesses the sensor data via a serial cable connection between the PC and the sensor. From there, the data could be read from the PC through a wireless internet connection already in place on site.

When deciding what software tools to use for developing the components of the system, we had several options to consider. Early in the development process, we each researched the available options for our assigned components, and then discussed them at our meetings. We made suggestions and helped determine compromises for each section.

For the GUI, we considered using either Java or GTKmm. We knew that our program was to be written in C++, so linking a Java GUI to the C++ back end would be a challenge. It would require writing dll wrappers to interface the java to the C++, which proved to be too cumbersome and time-consuming. On the other hand, the GTKmm API has a C++ interface, which eliminated the problem with linking the GUI to the main code.

The SRM is built upon open source code that was extended to provide discreet spectral capabilities. This original code was written in FORTRAN and formed a working radiative transfer model. It was based off arbitrarily defined polygons, but still produced accurate results. We modified the code to operate with our ray-tracing algorithm. The challenge was to combine it with our C++ code. We identified two plausible options: to rewrite the FORTRAN subroutines in C++, or create a C++ interface to make the subroutine calls. Rewriting the code would require a significant amount of work for coding, as well as revalidation of the correctness of the solar model. FORTRAN subroutines cannot return a value, but we solved this problem by passing variables by reference through the interface. Since interfacing the Fortran with C++ functions required less work and was less prone to bugs, we decided to preserve the Fortran code. Another issue the team faced was what development environment to use. The visualization component uses VTK, which is designed to compile under Cygwin. The GUI was originally developed with Visual Studio .NET, but was switched to Cygwin to accommodate integration with the other components of the system.

Conclusion

Summary

Researchers interested in the propagation of solar radiation now have a powerful tool at their disposal. SunRay is currently a functional prototype, implementing all of the features necessary to run a solar exposure simulation on any 3D mesh presented in 3DStudio or GTS format. While the software exists as a working system, it is far from complete. Currently, the team is continuing work on testing and debugging the prototype, as well as further developing the interface between the pyranometer network and the software.

There will always be areas in which the system could be improved, and in which more efficient analysis of the data could be facilitated. The utility of the prototype as it exists, however, is significant. We believe the SunRay system makes a contribution to the study and understanding of solar radiation and its relation to terrestrial life. The system provides data that has never before been available to the scientific community through spectrum-specific data results, calculations of total body burden, and accurate predictions of the effects of changing atmospheric conditions on life.

Recommendations for the Future

Throughout the design, development and testing of the system, ideas for future system enhancements were repeatedly discussed. For the sake of time, however, limits had to be placed on additions to the requirements. Some of these additional features were deemed feasible, and were integrated into the requirements, but the remaining additions had to be logged as possibilities for the future. For example, several potential extensions to the handling of 3D forms could provide much greater flexibility.

During all stages of work on this project, we kept in sight the goal of creating easily extensible software in order to allow and encourage enhancement in the future. Plentiful possibilities for extension exist that could provide meaningful data to the scientific community as well as the layperson. Adding the capability to incorporate an animation sequence into the ray-tracing process would provide a more accurate representation of the exposure resulting from specific activities involving repetitive motion or changing positions. Furthermore, the reflective properties of materials could be considered in order to gauge the contribution of water, ice, or snow toward irradiance.

Other future enhancements may be more practically tied to the preservation of health. For instance, SunRay could be extended to predict the daily dosage of solar radiation under hypothetical environmental changes, or to estimate the risk of cataracts over an extended period of exposure. As another extension, SunRay could model the attenuation of sun exposure due to clothing and indicate the level of sun protection offered by certain types of clothing or sunscreen. Such information could be quite helpful in providing clues to how we can better protect ourselves against harmful amounts of radiation. Although many of these applications directly benefit humans, animals are equally vulnerable to solar radiation. Efforts to mitigate sun exposure, such as preserving forests and reducing agents that deplete the ozone layer, will benefit humans, plants and animals alike.

References

1. CIESIN Ozone Home Page (2006): http://www.ciesin.org/TG/OZ/oz-home.html
2. INTERSUN – The Global UV Project: http://www.who.int/docstore/peh-uv/pub/who-ehg-95-16.htm
3. USDA UVB Radiation Monitoring Program Home Page (2006): http://uvb.nrel.colostate.edu/
4. WHO – Magnitude and Causes of Visual Impairment (2004): http://www.who.int/mediacentre/factsheets/fs282/en/index.html
5. Caldwell, M.M. and Flint, S.D., Stratospheric ozone reduction, solar UV-B radiation and terrestrial ecosystems, Climate Change, 28, 375-394, 1994.
6. Diffey B, Kerwin M, Davis A. Anatomical Distribution of Sunlight, British Journal of Dermatology, 97 (4): 407-410 1977
7. Streicher, John J., Culverhouse, William C., Dulberg, Martin S., Fornaro, Robert J. Modeling the Anatomical Distribution of Sunlight. Photochemistry and Photobiology 2004 79: 40-47
8. Pharr, M. and G. Humphreys. P*hysically Based Rendering: From Theory to Implementation*. Elsevier Inc. 2004. pp. 198-220.
9. The libRadtran homepage (2006): http://www.libradtran.org/
10. The Stanford 3D Scanning Repository (2006): http://graphics.stanford.edu/data/3Dscanrep/
11. GTS – Sample Files (2006): http://gts.sourceforge.net/samples.html
12. Ken Schwaber and Mike Beedle, Agile Software Development with Scrum (Prentice Hall, 2001).