

Thermal performance analysis of heat shielding under post-nuclear explosion conditions

By Shazee Avaes Shah

A thesis submitted in partial fulfilment of the
requirements for the degree of

MSc Scientific and Engineering Software Technology

University of Greenwich
Department of Computing and Mathematical Sciences

October 2002

The University of Greenwich

Abstract

**Thermal performance analysis of heat shielding
under post-nuclear explosion conditions**

By Shazee Avaes Shah

Abstract: We design and develop a software package that will investigate the operating thermal temperature of different types of heat shielding placed in a region exposed to a post-nuclear blast. The thermal operating margins of the heat shielding will determine the success or failure of the candidate materials considered.

Contents

<u>1</u>	<u>INTRODUCTION</u>	<u>1-2</u>
<u>2</u>	<u>THEORETICAL CONSIDERATIONS</u>	<u>2-6</u>
<u>3</u>	<u>SOFTWARE DESIGN: OPTIMUS PRIME</u>	<u>3-38</u>
<u>4</u>	<u>EXPERIMENTAL PROCEDURE AND RESULTS</u>	<u>4-54</u>
<u>5</u>	<u>CONCLUSION</u>	<u>5-61</u>
<u>6</u>	<u>ACKNOWLEDGEMENTS</u>	<u>6-63</u>
<u>7</u>	<u>REFERENCES</u>	<u>7-63</u>

1 Introduction

In this project the thermal performance of various materials used for heat shielding, is investigated under extreme temperatures generated by a nuclear explosion. The purpose of the heat shielding, is that it will be used for military ground vehicles that are designed to operate near the nuclear blast region to collate scientific data on hydrodynamic effects on the environment caused by the blast. These vehicles will be unmanned, due to environmental extremities, such as high radiation levels that cause permanent biological damage as well as extremely high temperature conditions during the blast. The unmanned vehicles are being designed by the military to be deployed as near to the region of interest as possible. These military vehicles are to house extremely complex and expensive computer equipment worth millions of pounds. It is therefore imperative that the best heat shielding be used on the vehicle to withstand the extreme conditions that it will be deployed in. The complex computer equipment within the vehicle can only operate within given temperature margins, the upper limit being $120^{\circ}\text{C} \pm 10^{\circ}\text{C}$; any higher than this temperature will cause the system to overheat and meltdown. Physicists and material engineers at the research facility need to investigate the optimum heat shielding material needed to protect the equipment, but there are three main problems here that need to be addressed:

- 1). Live nuclear testing will require numerous tests to be performed in order to account for different experimental parameters. Such parameters include variations in material thickness, thermal properties and relative positioning from the point of blast of the heat shielding. More importantly, the variations in the thermal energy output of the nuclear device need to be accounted for; controlling the output energy of the bomb (yield) is extremely difficult even with the current level of nuclear weapons technology
- 2). Live nuclear testing will require numerous tests on various prototypes. With scarce and costly fissile material to select various desired test yield outputs. The fissile material can be reclaimed after each test of course (as long as it is not accidentally dispersed in an overshoot), but this takes time and either requires a large inventory of fissile material, or a very slow test program.
- 3). Why not test the shielding physically in the lab? Well, we need to accurately determine the ambient temperature where the shielding will be placed in the region of the blast, thus we cannot simply apply high temperatures to the shielding without accurate knowledge of the temperatures involved. Misinformation of the temperatures could cause will cause false outcomes and hence possible design failures in the prototype.

The only other alternative is computer modelling: cross over the whole problem onto a computer-modelling program. This will eliminate the physical component of the experimental process giving the scientists unlimited runs of the experiment without baring the high costs in logistics of large amounts of time and money. There are many different modelling programs on the market today, also many different solvers for larger complex mathematical problems. This would require the research scientists at the military labs to overcome the steep learning curve of understanding the complex subject of Computational Fluid Dynamics (CFD), mathematical modelling and some level of computer programming literacy. Commercially available software packages such as Pheonics and Ansys that perform these types of calculations would be advantageous, but these off-the-shelf packages are designed to solve generally all types CFD problems in 2/3D. The portability factor also needs to be addressed: How user friendly is these packages? Do we need to re-train new research staff to understand and efficiently operate the CFD package? How much time investment is needed to implement such a solution? How financially feasible is the CFD packages in terms of licensing and maintenance? How easy would it be to implement changes into the design criteria without having to outsource professional bodies? How portable are the output results; can they be readily imported to other packages? These are important factors to be considered when implementing the solution. CFD packages are very complex to set-up and use, but on the advantageous side, they give accurate results depending on the level of detail the user requires. For a better quality result we would refine the resolution of the mesh by setting the computational parameters in the package, but this would come at the cost of computational overheads such as processing time and memory usage (as there will be more points to calculate for on the grid).

So what would be a good alternative to this dilemma of off-the-shelf CFD packages? The proposed solution is to develop a custom-built system that would automate the whole process and have the following design features:

- 1). Intuitive user friendly interface for easy of use, and non-confusing irrelevant functions which are found in today's CFD packages.
- 2). Problem-specific solution. The system would be designed and developed around the user requirements making it fully functional to perform its designated job. In contrast, we have the generic CFD packages, which are designed to solve all types of problems, hence leaving it with a lot of redundant non-relevant functions.

- 3). Flexible programming model that would have scope for possible future implementations such as parallel algorithmic solvers or large scale migration to higher end systems
- 4). Lesser learning curve than that of conventional CFD packages. New members of the research team should be able to use the software without much specialist training. The software system is streamlined to solve specific problems directly linked with their line of research, which should make it easy for them to understand without ambiguities.
- 5). N-tiered model, i.e. client-server model to allow for greater data-to-system independency. With commercial CFD packages we have the data and program coalesced into one system. Any modifications in one part of the code would require all the other associated modules to be updated.
- 6). Platform independence. If the system is to be run on more powerful number crunching hardware, then the code whole system should not be bound to one particular platform, such as Intel's x86 platform.
- 7). Operating system independence, so that it would be easily migrated and integrated across to other OS platforms with little or no code modifications or complete cross engineering using a different programming language specific to the new platform.
- 8). The system has to perform the experiment in its entirety, i.e. without any third-party program interventions to completely obtain the final results. The results generated from the system have to be presented in a meaningful manner, such as graphical images and charts.
- 9). Accuracy of results is paramount, as the whole system is developed to replace the real nuclear tests. Thus scientific quality of the data values generated by the system has to be as close to the true value (if performed experimentally) as possible. Validation cannot be performed unless we set off a real nuclear weapon with the prototype in place within the region, this will prove difficult due to financial restrictions, but the prototype heat shielding can be developed and tested in a lab under extreme temperatures near to those calculated by the program.

The objective of this report is to test to see which heat shielding material will be best suited to perform under post nuclear temperatures for given parameters. In order to undertake this experiment we need to design and develop scientific software that would perform the thermal analysis test to see if a certain material will succeed in meeting its design criteria.

To outline remainder of the report:

Chapter 2 Theoretical Considerations: In this chapter the scientific and mathematical foundations needed to develop the software system and perform the experiment are explained in detail. The first section of this chapter details the physics of the nuclear weapon and its effects on the environment. It is from this information the necessary energy output of the nuclear weapon is obtained, as well as assumptions made for the physics model used in the software. The next section introduces the thermal engineering specifications for the heat shielding material, and the candidate materials chosen for this investigation. The last section of the chapter shows how CFD numerical techniques (namely finite volume method) can be used to model the problem on computer; thermal convection - which corresponds to the heat transferred to the environment due to the nuclear explosion and thermal diffusion - corresponding to the ambient heat conducted across cross section of the heat shielding material. The groundwork for the main algorithms used to solve the CFD problems are laid out in this section, that are incorporated into the system.

Chapter 3 Software Design [Optimus Prime]: This chapter takes the reader through the analysis and design of the application, Optimus Prime. Using the Object Oriented approach in UML, the system requirements are taken down and the use case diagrams fleshed out. The variables to be stored on database are isolated and implemented into the database design. The conceptual class diagrams that are derived from the system requirements specifications are given using Rational Rose UML modelling tool. The sequence diagrams for the two main use cases, namely pre-processing and run experiment, displaying the interactions between the classes are shown. The Pre-processing wizard and Post-Processing design in Java are explained; as well the problems faced during the design stage are discussed. The implementation of key algorithms showing the post-processing design features is given. Finally the experiment options and additional features of Optimus Prime are explained at the end of this chapter.

Chapter 4 Experiment & Results: In this chapter a functional analysis is performed on the software to validate the physics engine and evaluate the computational performance. Increasing the workload, namely the mesh resolution, to the maximum limit and execution time recorded and logged onto a graph, is the basis of the computational tests. The physics engine is tested by setting the experimental variables (in pre-processing) to an arbitrary value, and using it as template to vary each physical attribute (such as specific heat capacity of air) from the upper to lower limit. The experiment is executed at each extreme to observe the changes on a thermograph (taken from Optimus Prime), then analysed to see if they are physically correct.

Chapter 5 Conclusions: In this final chapter the failures and successes of Optimus prime are discussed in detail and the improvements that could be made. Scalability of the system and recommendations for future research into this study are given at the end of this section.

2 Theoretical Considerations

2.1 The Nuclear Weapon

2.1.1 The physics of nuclear weapons

In this section of the chapter the energy output of the nuclear weapon is determined, as well as the types of nuclear weapons. This data will be used to build a database for the software system, allowing the user to select the nuclear weapon for a desired yield. It is the aim of this chapter to introduce the reader to the basic concepts and terminologies used in nuclear weapons physics.

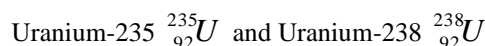
What is a nuclear explosion?

A nuclear explosion is a sudden release of large amounts of energy within a limited space; this system later reaches stability after the newly introduced energy from the explosion is dissipated into the environment. The basic laws of thermodynamics referring to the conservation of energy require that energy must be released when a system is converted to another of greater stability, i.e., one containing less energy.

Energy released in a nuclear explosion is not produced by chemical reactions but a nuclear reaction called fission and fusion, in which fundamental changes occur in the nucleus of the atom (known as the nuclei) of the reacting material. In these nuclear reactions mass is actually converted to energy, and the amount of energy produced is many orders of magnitude greater than that available from chemical reactions. It is important to understand the basic principles of atomic structure and nuclear reactions in order to understand the nuclear reactions given off from these weapons.

2.1.1.1 Isotopes

Atoms of different elements have different numbers of protons in their nuclei. The term atomic number (denoted by A) describes the number of protons in a nuclei of an element, and Atomic Mass Number (denoted by Z) is the total number of protons + neutrons in the nuclei of an atom. Although all the nuclei of a given element will have the same atomic number, they may have different atomic masses because they may contain different numbers of neutrons.¹ Normally this does not affect the stability or chemical properties of elements, this is because the number of protons has not changed, but it affects the nuclear stability of different atoms. Isotopes are atomic elements (species) with identical atomic numbers (A) but different atomic mass numbers. For example, let us look at the isotopes of Uranium -



- Both contain the same number of protons ($A = 92$ {atomic number}), but the differing number of neutrons in the nucleus accounts for difference in atomic mass numbers, 235 and 238. There are two kinds of isotopes: stable or unstable. Let us now look at the isotopes of 83 stable elements (NB: elements are also known as naturally occurring stable nuclides). If we plot a graph of the number of neutrons ($A-Z$) against the number of protons of these 83 elements a linear correlation can be identified. Referring to the graph in Figure 2, we see that the stable isotopes of elements have very definite ratios of neutrons to protons in their nuclei. As atomic mass numbers increase, the ratio of neutrons to protons increases according to a definite pattern. If isotopes vary from this pattern, they are relatively unstable.

2.1.1.2 Mass defect

¹ Atomic notations: ${}_Z^AX$; X is the name of the chemical element symbol (e.g. Al for aluminium). A is the atomic mass number, i.e. the number of protons + neutrons in the nucleus. Z is the atomic number; number of protons in the nucleus. For brevity ${}_Z^AX$ notation is sometimes used, for example Uranium-235 would be denoted as ${}^{235}\text{U}$

Atomic mass unit: Common units of mass, such as grams, are much too large to conveniently describe the mass of an atomic nucleus or any of its constituent parts. To solve this problem a new unit was defined: the atomic mass unit (amu). The atomic mass unit is a relative unit defined arbitrarily by assigning a mass of 12 amu to the neutral atom carbon-12, the common isotope of carbon. One atomic mass unit equals 1.66×10^{-24} grams. We then find the atomic masses of the nucleons to be:

Proton mass: 1.00727 amu
 Neutron mass: 1.00867 amu
 Electron mass: 0.00055 amu

So, if we were to weigh the atom, we would expect common sense to say that it would be the sum of the masses of the constituent particles (amu of atom = amu of protons + amu of neutrons + amu of electrons). But a survey of the atomic masses shows that the atomic mass is less than the sum of its constituent particles in the free state!

To account for this difference in mass, the principle of the equivalence of mass and energy, derived from special relativity is used. If ΔM is the decrease in mass when a number of protons, neutrons and electrons combine to form an atom, then the conservation of energy states that the missing amount of energy is equal to: $\Delta E = c^2 \Delta M$ is released in the process; c is the velocity of light in a vacuum. The difference in mass, ΔM , is called the *mass defect*: i.e. the amount of mass that would be converted into energy if a particular atom were to be assembled from the requisite numbers of protons, neutrons and electrons. The same amount of energy would be needed to break the atom into its constituent particles, and the energy equivalent of the mass defect is thus a measure of the *binding energy* of the nucleus. For example, the measured mass of the isotope fluorine-19 atom is 18.99840 amu, while the sum of the masses calculated for the individual particles of that atom is 19.15708 amu. The difference of 0.15868 amu between the measured and calculated mass of the fluorine-19 atom is defined as the mass defect.

2.1.1.3 Nuclear fission

Nuclear fission occurs when the nuclei of certain isotopes of very heavy elements capture neutrons. Figure 1 illustrates this process; the uranium-235 isotope captures the inbound neutron. The neutron introduces a small amount of energy to the nucleus of uranium, which causes the uranium nucleus to enter an unstable state causing the uranium nucleus to split into roughly two equal pieces. In doing so, there is a release of large amounts of energy (180 MeV of immediately available energy; Note: 1 electronvolt [1eV] = 1.60210×10^{-19} Joules [J]) and an average of 2.52 new neutrons (an average of 2.95 for Pu-239). Now, if on average one neutron (from each fission process) is captured and successfully causes other uranium nucleuses to undergo the same fission process, then a self-sustaining chain reaction is produced, hence the number of neutrons and the rate of energy production will increase exponentially with time.

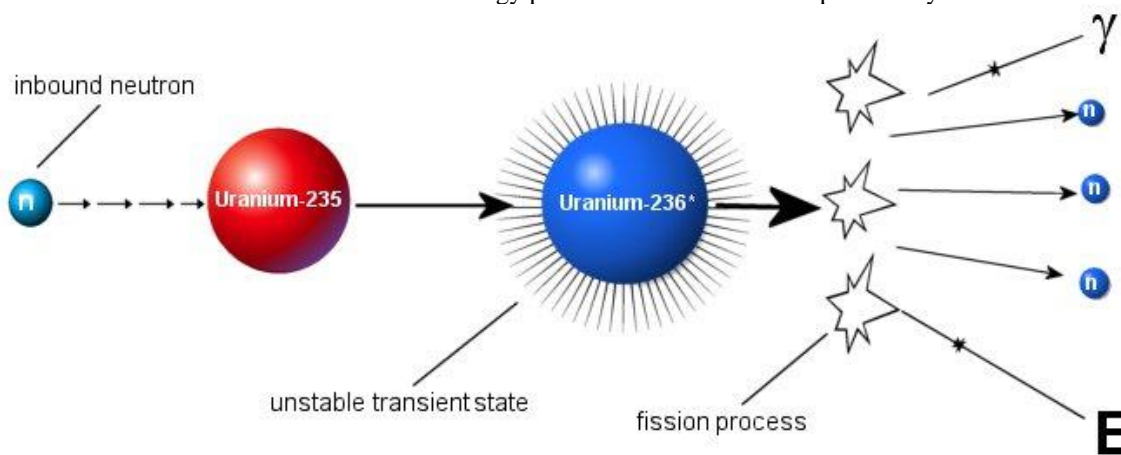


Figure 1 The fission process

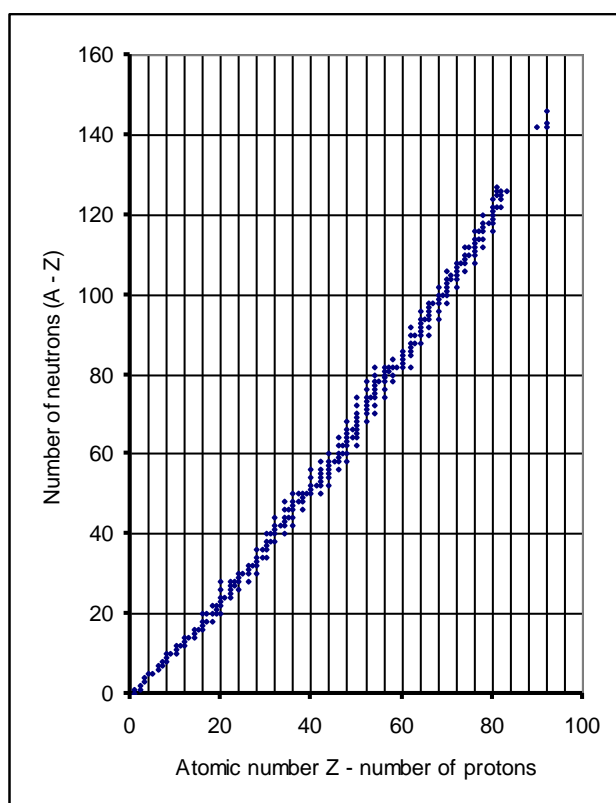


Figure 2 Graph showing the stable nuclides

captured, the energy released by rearrangement exceeds it. The nucleus is then no longer stable and must either shed the excess energy, or split into two pieces. Since fission occurs regardless of the neutron's kinetic energy (i.e. no extra energy from its motion is needed to disrupt the nucleus), this is called "slow fission". On the other hand, when the abundant isotope uranium-238 captures a neutron it still has a binding energy insufficiency of 1 MeV after internal rearrangement. If it captures a neutron with a kinetic energy exceeding 1 MeV, then this energy plus the energy released by rearrangement can overcome the binding energy and cause fission. Since a fast neutron with a large kinetic energy is required, this is called "fast fission".

The slow fissionable isotopes have high neutron fission cross-sections for neutrons of all energies, while having low cross-sections for absorption. Fast fissionable isotopes have zero fission cross-sections below a certain threshold (1 MeV for U-238), but the cross-sections climb quickly above the threshold. Generally though, slow-fissionable isotopes are more fissionable than fast-fissionable isotopes for neutrons of all energies. Referring back to figure 2 we see that the ratio of neutrons ($A-Z$) to protons (Z) in an atomic nucleus increases with the element's atomic number. Heavier elements require relatively more neutrons to stabilize the nucleus. When the nucleus of a heavy element like uranium (Z , atomic number 92) is split the fragments, having lower atomic numbers, will tend to have excess neutrons. The excited fragments shed these neutrons very rapidly. More neutrons are produced on average than are consumed in fission. Fission is a statistical process. The nucleus rarely splits into pieces with nearly the same mass and atomic number. Instead both the size and atomic numbers of the fragments have a Gaussian distribution around two means (one for the lighter fragment around 95, one for the heavier around 135). Similarly, the number of neutrons produced varies from zero to six or more, and their kinetic energy varies from 0.5 MeV to more than 4 MeV, the most probable energy is 0.75 MeV, the average (and median) is 2 MeV. A breakdown of the energy released by fission is given below in Table 1.

Fission products	Energy / MeV	Error / MeV
Kinetic energy of fission fragments	165	± 5
Gamma radiation (γ)	7	± 1
Beta particles from product decay (β)	7	± 1
Kinetic energy of neutrons	5	± 0.5
Neutrinos from product decay	10	-
Gamma rays from product decay	6	± 1
Total	200	± 6

Two conditions must be met before fission can be used to create powerful explosions:

- 1) The number of neutrons lost to fission (from non-fission producing neutron captures, or escape from the fissionable mass) must be kept low
- 2) The speed with which the chain reaction proceeds must be very fast. A fission bomb is in a race with itself: to successfully fission most of the material in the bomb before it blows itself apart. The degree to which a bomb design succeeds in this race determines its efficiency. A poorly designed or malfunctioning bomb may "fizzle" and release only a tiny fraction of its potential energy.

The stability of an atomic nucleus is determined by its binding energy - the amount of energy required to disrupt it. Any time a neutron or proton is captured by an atomic nucleus, the nucleus rearranges its structure. If energy is released by the rearrangement, the binding energy decreases. If energy is absorbed, the binding energy increases. The isotopes important for the large-scale release of energy through fission are uranium-235 (U-235), plutonium-239 (Pu-239), and uranium-233 (U-233). The binding energy of these three isotopes is so low that when a neutron is

Table 1 Total energy released during a fission process

All of the kinetic energy is released to the environment instantly, as are most of the instantaneous gamma rays. The unstable fission products release their decay energies at varying rates, some almost immediately. The net result is that about 180 MeV is actually available to generate nuclear explosions; the remainder of the decay energy shows up over time as fallout (or is carried away by the virtually undetectable neutrinos).

2.1.1.4 Nuclear Fusion

Fusion reactions can be seen as the opposite to fission reactions in that it involves the process of coalescing two lighter nuclei to form one heavier nucleus, contrasted with the breaking up of the nucleus into smaller nuclei. Fusion reactions, also known as thermonuclear reactions, will only take place when two nuclei have to be brought close together with enough energy to overcome the strong nuclear force. This force binds the protons and neutrons together in the nucleus of the atom; it overpowers the electrostatic forces of repulsion between charges of the same sign. There are two conditions needed in order for fusion to take place with a large number of nuclei:

- 1). High temperatures to accelerate the nuclei in order that they collide with each other with sufficient energy
- 2). High pressure density to increase the probability of interaction

The rates of all fusion reactions are affected by both temperature and density. The hotter and denser the fusion fuel, the faster the fusion "burn". The only practical way to obtain the temperatures and pressures required is by means of a fission explosion. Consequently, weapons with fusion components must contain a basic fission component. The energy released in the explosion of a fission-fusion weapon originates in approximately equal amounts from the fission and fusion processes. Incidentally, the fusion reactions that occur in stars are not the same as the ones that occur in thermonuclear weapons or laboratory fusion reactors. The somewhat complex catalysed fusion cycle in stars that converts light hydrogen (protium) into helium is extremely slow, which is why the lifetime of the Sun is measured in billions of years. The fusion reactions used in bombs and prospective power plant designs are simple, and extremely fast - which is essential since the fuel must be fully consumed within microseconds. These reactions thus are based on the same general principles as stellar fusion, but are completely different in detail.

2.1.1.5 Fusion reactions in thermonuclear weapons

The most important fusion reactions for thermonuclear weapons are outlined in table 2. Note: D and T stand for deuteron or deuterium (H-2), and triton or tritium (H-3) respectively.

1) D + T -> He-4 + n + 17.588 MeV
2) D + D -> He-3 + n + 3.268 MeV
3) D + D -> T + p + 4.03 MeV
4) He-3 + D -> He-4 + p + 18.34 MeV
5) Li-6 + n -> T + He-4 + 4.78 MeV
6) Li-7 + n -> T + He-4 + n - 2.47 MeV

Table 2 Fusion reactions used in thermonuclear weapons

The neutron produced in reaction 1 is extremely energetic; it carries away 14.06 MeV of the reaction energy, the alpha particle (He-4 nucleus) only 3.52 MeV.

The neutron produced in reaction 2 has energy of only 2.45 MeV (similar to the faster fission neutrons), with the He-3 carrying 0.82 MeV. The division of energy in reaction 3 is 1.01 MeV for the triton, and 3.03 MeV for the proton. The two D+D reactions are equally likely and each will occur half the time.

In reaction 4 the alpha particle carries off 3.67 MeV, the proton 14.67 MeV.

Reactions 5 and 6 are not thermonuclear reactions, strictly speaking. They are neutronic reactions and do not require heat or pressure, just neutrons in the correct energy range. This distinction is usually ignored in the literature about nuclear weapons however. The Li-6 + n reaction requires neutrons with energies in the low MeV range or below. The Li-7 + n reaction is only significant when the energies are above 4 MeV.

2.1.1.6 Critical Mass

In fission-type nuclear explosions, there must be enough material present and in the right configuration so that successive generations of neutrons can cause equal or increased numbers of fissions. This amount of fissile material that

is capable of sustaining a continuous or chain reaction is termed a critical mass. Thus, the minimum mass of a fissile material that will sustain a chain reaction is known as the critical mass.

There are three types of fission chain reactions:

- **Sub critical chain reaction** - In this reaction, the number neutrons produced by each generation decreases, and hence the fission reaction dies out.
- **Critical chain reaction** – The number of neutrons produced in this reaction remains constant throughout the succeeding generations
- **Supercritical chain reaction** – The number of neutrons increases exponentially in the succeeding generations.

To produce a nuclear explosion, a weapon must contain an amount of uranium or plutonium that exceeds the mass necessary to support a critical chain reaction, i.e. a supercritical mass of fissionable material is required. Several methods can be used to make a mass of fissionable material supercritical:

- 1) The active material can be purified to eliminate unwanted chemical impurities that might otherwise absorb neutrons.
- 2) The fissionable material can be fashioned into the most efficient shape. A spherical shape, for example, may be employed to provide the greatest volume with the least surface area to reduce the loss of neutrons.
- 3) Fissionable material can be enriched, i.e. the amount of ^{235}U as compared to ^{238}U can be increased.
- 4) Neutrons that have escaped the active material can be reflected back by using suitable materials as reflectors. Reflectors, used as tampers, can also physically delay the expansion of the exploding material allowing more fission to occur thereby resulting in an increase in explosive energy.

A nuclear weapon should not contain fissionable material as large a critical mass. This is due to the fact that the fissionable material is exposed to naturally occurring neutrons present in the atmosphere, which could cause the fissionable material to melt or possibly explode! It is important that the following conditions be met before constructing a fissionable weapon:

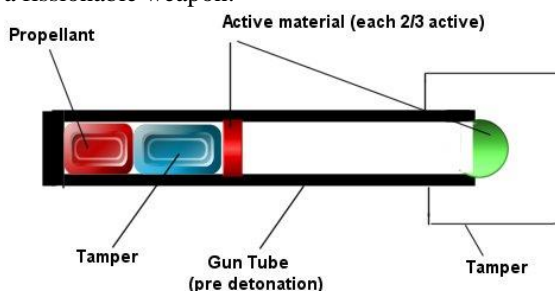


Figure 3a Gun Assembly

- 1) Sustaining the fissionable material in a sub critical state pre detonation phase
- 2) Instantiating the fissionable material into a supercritical mass while keeping it free of neutrons
- 3) Introducing neutrons into the critical mass when it is at maximum super criticality
- 4) Keeping the mass together until a substantial portion of the material has fissioned

The key to achieving objectives 1 and 2 is revealed by the fact that the critical mass (or supercritical mass) of a fissionable material is inversely proportional to the square of its density.

By contriving a sub critical arrangement of fissionable material whose average density can be rapidly increased, we can bring about the sudden large increase in reactivity needed to create a powerful explosion. As a general guide, a suitable highly supercritical mass needs to be at least three times heavier than a mass of equal density and shape that is merely critical. At the time of detonation, a method must be employed to make the mass supercritical. There are two main methods for converting the fissionable sub critical mass into a supercritical mass:

Gun Assembly Principle: Two pieces of fissionable material, each less than a critical mass, are brought together very rapidly to form a single supercritical one. This gun-type assembly may be achieved in a tubular device in which a high explosive is used to blow one sub critical piece of fissionable material from one end of the tube into another sub critical piece held at the opposite end of the tube as in Figure 3a.

Advantages: simple and foolproof design

Disadvantages: 1). lack of compression; requires large amounts of fissionable material which leads to low efficiency.

2). Only ^{235}U (or ^{233}U possibly) can be used due to the slow insertion speed. 3). The weight and length of the gun barrel makes the weapon heavy and fairly long.

Implosion Assembly Principle: Explosives placed around the fissile core cause a smooth, symmetrical implosive shockwave. These shockwaves compress the fissionable material core instantiating a fission reaction taking it into supercriticality. The implosion raises the density to the point of supercriticality. As shown in Figure 3b.

The high explosives generate pressures of 400×10^3 atm.

Advantages: 1). High insertion speed – allows isotopes with high spontaneous fission rates to be used (i.e. plutonium) 2). High density achieved \Rightarrow very efficient bomb: less material is required. 3). Potential for lightweight designs: only several kilograms of explosives are needed to compress the core.

We have covered the basic principles involved in nuclear weapons' physics. In the next section we take a look at the types of weapons and their energy outputs. From this data we can develop a database of nuclear

2.1.2 Classification of nuclear weapons

In this section of the chapter the types of nuclear weapons that are used in the real life experiment are outlined. The information about the types of weapons outlined in this chapter will be implemented into the software system.

2.1.2.1 Units of explosive energy - ton

When measuring the explosive energy output, or yield, of a nuclear weapon the term "ton" and its metric extensions (kiloton, megaton, etc.) are used as units. This should not be confused for the metric tonne (= 1000 kg) used in weight. The units of explosive energy (megatons, kilotons, tons, depending on yield) are derived from attempts to compare the explosive force of a bomb to conventional explosives; the original intention was to equate it with tons of trinitrotoluene (TNT) - a workhorse military explosive. The problem arose to which "tons" are we trying to compare? And the explosive force of TNT is not exactly a universal constant. The energy release is affected by such things as charge density, degree of confinement, temperature, and the reference end state of the explosion products. Energy outputs ranging over 980-1100 calories/g are reported. Physicists employ the SI units system, so we need to convert the "ton" over to Joules as follows:

$$\begin{aligned} \text{ton} &= \text{one unit of explosive energy/output/yield} \\ 1 \text{ kiloton} &= 10^{12} \text{ calories} = 4.186 \times 10^{12} \text{ J} \end{aligned}$$

Note that the metric definition of kiloton refers to ALL of the energy immediately released by the device, regardless of form. Although chemical explosives release essentially all of their energy as kinetic or blast energy, only part of the energy in a nuclear explosion is released in this form (though under most conditions, it is the major part). Thus a kiloton nuclear explosion actually has significantly less blast effect than a kiloton chemical explosion.

2.1.2.2 Pure fission weapons: Gun Assembly

Type	Fission reaction
Detonation	Rapidly assembling a sub critical configuration of fissile material into one that is highly supercritical using gun assembly design.
Efficiency	$\approx 1.4\%$
Test	"Gadget": Japan 6 1945 {"Little Boy"} / Hiroshima and Nagasaki
Materials	Plutonium or enriched uranium

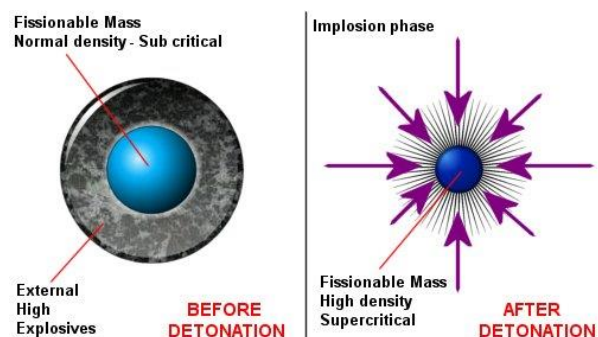


Figure 3b Implosion assembly

Disadvantages	<p>Larger bombs implies more fissionable material</p> <ol style="list-style-type: none"> (1) Becomes increasingly difficult to maintain as sub critical mass before detonation (2) Makes it harder to assemble into a high efficiency supercritical mass before stray neutrons cause pre-detonation
Largest known bomb	500kt: "Ivy King" 15 Nov 1952

2.1.2.2 Combined fission – fusion weapons: Implosion Assembly

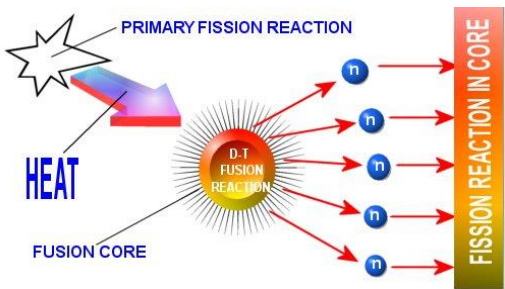
Type	Boosted fission weapons
Materials	Few grams of deuterium-tritium gas mixture centred in the fissile core. Cryogenic deuterium-tritium (D-T) can also be used.
Reaction	 <p>Referring to Figure 4, When the bomb core undergoes enough fission it becomes hot enough to ignite the D-T fusion reaction in the fusion core. This secondary reaction produces an intense burst of high-energy neutrons that causes a corresponding intense burst of fission in the core. This process allows a much higher percentage of material to undergo fission before the assembly blows apart.</p> <p>No more than 20% in an average size fission bomb will split before the reaction ends.</p>
Efficiency	≈ 20% fissile material undergoes reaction.
Fission yield	(Boosted) 100% implies an unboosted 20kt bomb can become a 40kt bomb via boosting by accelerating the fission process.
Fusion yield	Makes up ≈ 1% of the total bomb's yield, which is nearly undetectable as compared to a pure fission bomb.
Fusion core	Apart from the D-T gas/liquid booster, there is deuterium gas only and lithium deuteride/tritide
Test	"Greenhouse Item" 45.5kt: Enewetak 24 May 1951
Disadvantages	<ol style="list-style-type: none"> (1) Tritium is expensive, but only a few grams is required (2) Tritium decays at 5.5% per year
Advantages	<ol style="list-style-type: none"> (3) Reducing the fissionable material ⇒ reduced risk of pre-detonation (4) Increases the yield

Figure 4 Boosted fission design

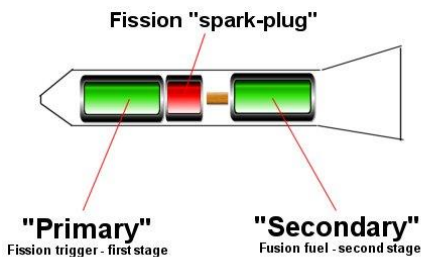
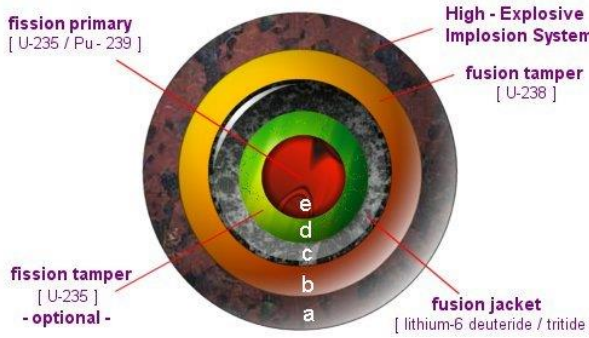
Type	Staged Radiation Implosion Weapons ("Teller-Ulam") Fission-fusion or fission-fusion-fission design
Material	Fusion of isotopes of pure deuterium, or varying mixtures of lithium 6 and 7 in the form of a compound with deuterium (lithium 6/7 deuteride).
Reaction	 <p>Referring to Figure 5:</p> <ol style="list-style-type: none"> 1). X-Rays from the primary trigger are used to compress the secondary trigger via radiation implosion 2). Secondary trigger is ignited by fission "spark-plug" at the center. 3). The energy produced by the secondary fusion stage can be used to ignite an even larger fusion third stage. 4). Multiple staging ⇒ creation of bombs of virtually unlimited size.
Modifications to increase yield	A fissionable jacket is placed around the fusion core in the secondary stage. This increases the overall yield of the weapon. This is achieved by using high-energy neutrons (or "fast" neutrons) generated by <u>fusion</u> to release energy through fissioning of the fissionable jacket placed around the fusion material in the secondary stage. The jacket is made of natural or


Figure 5 Two-stage fission-fusion design

	depleted uranium (U-238). Sometimes thorium is used in place of U-238. Most of the modern strategic nuclear weapons use highly enriched uranium as the jacket material.
“clean” bombs	Bombs that obtain a large majority of their total yield from fusion, where the percentage of material that undergoes fusion is as high as 97%
“dirty” bombs	Known as fission-fusion-fission bombs: They generate a large amount of fission fallout since fission accounts for most of their yield.
Test	“Bassoon” 3.5Mt ‘clean’ 3 stage device, 85% fusion material undergoes reaction “Tsar Bomb” 50Mt ‘dirty’ 3 Stage device, 97% fusion material undergoes reaction “Bassoon Prime” 25Mt ‘dirty’ device “Tsar Bomb” 100Mt ‘dirty’ fission-fusion-fission (largest bomb ever detonated) “Mike” 10.4Mt ‘clean’ 77% of which was fission “Zombie” 1.69Mt ‘clean’ two-stage pure fission device
Advantages	1). The natural stable isotopes used are vastly cheaper than the artificially made and radioactive tritium. 2). Large yields are attainable through fusion alone and enhanced through fission “jacketing” the secondary fusion core

Type	The Alarm Clock / Layer Cake Design (Sloika)	
Materials	Fusion primary: U-235 or Pu-239 Fission tamper: U-238 (Depleted uranium) Fusion jacket: Lithium-6 deuteride/tritide Outer fusion tamper: U-238 (Depleted uranium) High explosives for imploding the system	
Yield	Tritium doping made the “Alarm Clock” weapon achieve ten fold boost over the size of the trigger which gave a yield of 400kt	
Note	This design is a distinct class from the other weapons. This has a hybrid design; it has the features of a boosted fission device and a one-stage fission-fusion-fission bomb. This design was developed before the staged-thermonuclear designs.	
Reaction	<div></div> <p>Referring to Figure 6: <u>Phases of the explosion process</u></p> <ol style="list-style-type: none">1). Implosion bomb goes off (a)2). Fission primary reacts (e). Heats and compresses the fusion layer (c) to thermonuclear temperatures.3). Burst of fission neutrons initiates a coupled fission – fusion – fission chain reaction.4). Slower fission neutrons generate tritium from lithium-65). The tritium produced in phase 4) fuses with deuterium to produce very fast neutrons(c)6). These fast neutrons cause fast fission in the fusion tamper (b)7). Phase 6). Causes the breeding of more tritium releasing large amounts of energy	
Disadvantages	This design has 15-20% fusion efficiency, i.e. the amount of fusion material that undergoes fusion. Fusion fuel is quite inefficient and expensive.	
Test	“RDS-6s – Joe 4” 400kt, 10-fold boost over the trigger size “UK Orange Herald Small” 720kt, 300kt large trigger	

2.1.2.3 Neutron Bombs

Type	Enhanced Radiation (ER) Weapons
Brief	Small thermonuclear device, where the burst of high-energy neutrons generated by the fusion reaction is NOT absorbed inside the weapon, but allowed to escape. Neutrons are destructive, i.e. more penetrating than other types of radiation; neutrons can penetrate shielding which even γ -rays penetrates. “Enhanced radiation” is a burst of ionising radiation released at the moment of detonation, not any enhancement of residual radiation fallout.
Usage	Strategic anti-missile weapons Tactical weapons against armoured forces

Anti-missile	ER (Enhanced Radiation) was developed to protect U.S ICBM silos (Intercontinental Ballistic Missile) from incoming soviet warheads by damaging nuclear the main components of incoming warhead by an intense neutron flux.
Tactical neutron bombs	Used originally to kill soldiers protected by armor. Armored vehicles are extremely resistant to blast and heat produced by nuclear weapons. The lethal range of radiation determines the effective range of a nuclear weapon against a tank, although this is reduced by armor. ER warheads maximise the lethal range of a given yield of a nuclear weapon against armored targets.
Effects	<ol style="list-style-type: none"> (1) Rapid incapacitation of the target is achieved by administering radiation many times over the lethal dose (approx. 600 rads) (2) No effect noticed until several hours (3) Neutron bombs were originally designed to deliver 8000 rads for immediate incapacitation. 1kt ER warhead can do this to a T-72 tank crew at a range of 690m compared to 360m for a pure fission bomb. (4) Further, radiation doses: for a 600 rads dose 1100m (ER bomb) and 700m (fission bomb) and for unprotected soldiers (600 rads) 1350m (ER bomb) and 900m (fission bomb)
Secondary radiation	The amount from the flux can create secondary radiation of short-lived secondary radioactivity in the environment near the flux region from ground zero. The alloy steel used in armor can develop radioactivity that is dangerous for a period of 24-48 hours
Countermeasure armor	 <p>Special neutron absorbing armor has also been developed and deployed using boronated plastics and uses the vehicle fuel as a shield. The M1 tank employs 105mm depleted uranium shells, which can offset these improvements since it undergoes fast fission generating additional neutrons and becoming radioactive.</p>
Range effectiveness	Neutron ER-weapons are of short range due to absorption of neutron energy by the atmosphere. Neutron energy drops by a factor 10 every 500m, as well as effects of divergence spreading. Neutron bombs are known as the “Landlord” bomb because it leaves all the buildings and structures around it intact, while killing off all organic life. Friendly forces can use this bomb at close range due to its low yield and because the ER-warhead is designed to minimise the amount of fission energy.
Combat range	≈ 690m for a 1kt neutron bomb: will destroy or damage, to the point of un-usability, almost any civilian building
Deployment	Blanket bombing around the enemy forces
Materials	Deuterium – tritium (D-T) gas mixture. For 1kt yield: 12.5g tritium and 5g deuterium
Reaction	D-T reaction releases 80% of its energy as neutron kinetic energy Powerful 14.7 MeV neutron kinetic energy from a very small fission explosion: 250-450 tons
Disadvantage	D-T fuel: 1). Tritium is very expensive 2). Decay rate is 5.5% per year

2.1.2.4 Cobalt / Salted bombs

Type	Similar to fission-fusion-fission bombs but instead of a fissionable jacket around the 2 nd stage fusion material, a non-fissionable blanket of a specially chosen salting isotope is used.			
Materials	Parent isotopes	Natural abundance	Radioactive product	Half-life
	Colbalt-69	100%	Co-60	5.26 years
	Gold-197	100%	Au-198	2.697 years
	Tantalum-181	99.99%	Ta-182	115 days
	Zinc-64	48.89%	Zn-65	244 days
Reaction	Blanket captures escaping fusion neutrons to breed a radioactive isotope that maximises the fallout hazard from the weapon as opposed to generating additional explosive force, and dangerous fission fallout from fast fission (U-238).			
Fallout effects	Salting isotope used: <u>Gold</u> : short-term fallout (days) <u>Tantalum and zinc</u> : intermediate duration (months) <u>Cobalt</u> : long-term contamination (years)			

2.1.3 Effects on the environment by a nuclear explosion

In this section, we discuss the effects of setting one of the nuclear warheads off in the atmosphere. Location of the point of blast within the environment is very important, as the yield of the weapon determines the energy distribution profile.

2.1.3.1 General Effects

There are three distinct forms of energy released by a nuclear blast:

1. Thermal radiation
2. Nuclear radiation
3. Mechanical blast

The distribution of these energies depends on three major factors:

1. Yield of the nuclear weapon
2. Location of the point of detonation, i.e. the burst-point
3. Characteristics of the immediate environment

Low altitude atmospheric detonation of weapons in the kiloton yield range the energy distribution would be as that in Figure 7.

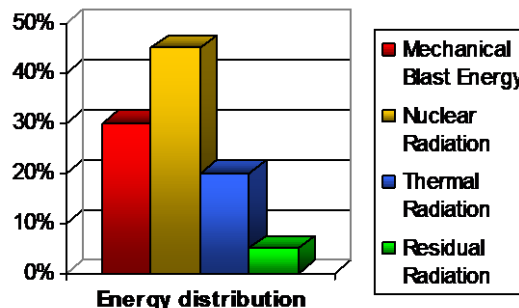


Figure 8 Enhanced radiation energy distribution

1. 50% Mechanical blast

2. 35% thermal radiation composed of infrared radiation, visible light, ultraviolet radiation and some soft x-ray

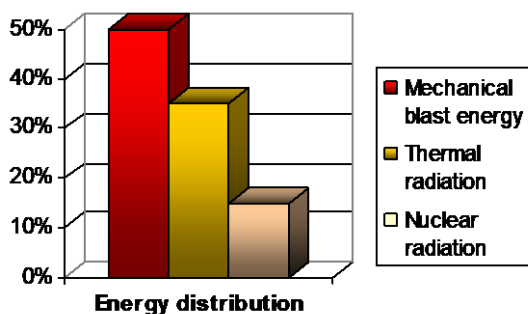
emissions

Figure 7 Energy distribution of nuclear weapons with yields in the kiloton range

3. 15% nuclear radiation composed of 5% as initial ionizing radiation with high level of neutrons, gamma rays emitted within the first minute after detonation, and 10% as residual nuclear radiation. Residual nuclear radiation is the hazard in fallout.

2.1.3.2 Fireball formation

Figure 8 shows the energy distribution for enhanced radiation weapons, i.e. weapons that have been modified to enhance the total percentage of radiation and reduce the percentage of mechanical blast and thermal energy given out at detonation by the warhead. When a nuclear weapon is detonated extreme amounts of energy is dissipated into the immediate atmosphere, reaching temperatures of tens of million degrees Kelvin around the blast area. Under these high temperatures non-fissioned isotopes from the warhead are atomised. The nuclear reaction from the weapon releases large amounts of electromagnetic radiation dominantly composed of soft X-Rays. Atmospheric detonation absorb most of the soft X-Rays within a few meters causing the surrounding atmosphere to heat to extremely high temperatures and form a brilliantly hot sphere of air and gaseous weapon residues, this is called the "fireball". Post detonation, The fireball expands rapidly and begins to rise at a velocity of 100m/s, a millisecond later the diameter of the fireball from a one megaton air burst is 150m, this increases to an upper limit of 2200m within 10 seconds! The initial rapid expansion of the fireball causes extreme compression in the surrounding atmosphere producing a powerful blast shockwave. The fireball emits large amounts of thermal radiation. There is a bright visible flash of light at the point of detonation and the bright glow of the fireball. It is the infrared component that causes the burns and incendiary effects. During the expansion the fireball loses heat to it's surroundings until it no longer emits high levels of thermal radiation. The upward



movement and cooling of the fireball shapes itself like the well known mushroom cloud. As the fireball cools down, the composite materials within the cloud condense to form a cloud of solid particles. An air-burst occurs immediately afterwards, this burst consists of condensed droplets of water which gives rise to the cloud-like appearance. In the case of a surface burst, this cloud would contain large quantities of dirt and other debris which are vapourised when the fireball touches the earth's surface. The dirt and debris become contaminated with radioisotopes generated by the explosion or activated by neutron radiation and fall back down to earth as fallout debris. After approximately 10 minutes the cloud stabilises, depending on the thermal output of the weapon and atmospheric conditions. The

nuclear cloud from a 1 Mt surface burst will stabilise at an altitude of over 20 km and will have a mean diameter of 35m. The cloud would then fall back down to earth as fallout debris.

2.1.3.3 Burst Classifications

There are four different types of burst classification, i.e. the height of detonation. It is this height that determines the relative effects of the blast, heat and nuclear radiation. They are as follows:

1). Air Bursts: Mid-air explosion of nuclear weapon at altitude below 30km. The purpose of an air-burst is so that the radiation and thermal exposure is creates optimal damage. There are no debris fallout hazards from air-bursts. The fission products of the warhead are generally dispersed over a large area of the globe. In the region of ground zero there may be a small area of neutron-induced activity, which could be hazardous to troops required to pass through the area. Tactical significance of an air-burst is that may be used against ground forces.

2). High Altitude Burst: Weapon is detonated at an altitude above 30km, so that the initial flash of soft X-Rays generated by the detonation dissipate thermal energy in a much larger volume of air molecules. Thus, the fireball is much larger and expands much more rapidly. The absorption distance for the ionizing radiation is much greater. Severe disruption to wireless communications occurs due to the significant ionisation of the ionosphere. An intense electromagnetic pulse (EMP) can cause significant damage to sophisticated electronic equipment.

3). Surface Burst: The weapon is detonated on/slightly above the earth's surface. The purpose being is so that the fireball actually touches the land or water surface. The area affected by the blast / thermal radiation / initial nuclear radiation will be less extensive, around ground zero, then an air burst of the same yield. In contrast with air bursts, local fallout can be a hazard over a much larger downwind area than that which is affected by blast and thermal radiation.

4). Subsurface Burst: The weapon is detonated beneath the earth's surface, land or water. Seismic cratering caused by the subsurface blast. If the burst does not penetrate the surface, there will be hazard from ground or water shock. If the burst is shallow enough to penetrate the surface then the blast / thermal / initial nuclear radiation effects will be present, but will be less than that of a surface blast of same yield.

2.1.3.4 Thermal radiation

The surface of the fireball emits large amounts of radiation across the visible, infrared and UV regions of the electromagnetic spectrum. This all occurs in the first minute of detonation. The greatest hazard from the thermal radiation is burns and eye injuries to exposed personnel, these injuries can occur even at the furthest point from the blast where initial nuclear radiation effects are minimal. The range of the thermal effect increases with the weapon's yield.

Propagation of thermal energy

A large percentage of the energy released is kinetic energy of the fission fragments and other products of the reaction. A millionth of a second after detonation, numerous inelastic collisions of these vaporised atoms give rise to plasma of extremely high temperatures of several tens of million degrees centigrade! Due to these extreme temperatures, large amounts of E-M radiation is emitted from the plasma, which is absorbed by the surrounding atmosphere, raising it to an extremely high temperature, this causes the air to emit secondary radiation of slightly lower energy. This complex process of absorption and emission of radiation is the fundamental mechanism which forms the fireball and accounts for its expansion. The distance between the point of emission and absorption is relatively long (called the mean free path), the initial expansion of the fireball is extremely rapid: faster than the outward motion of the gaseous material from the center of the burst, which causes the blast wave.

As the fireball expands, its temperature decreases and the transfer of energy by thermal radiation slows down. The separate blast wave, which is behind it, catches up and supersedes the fireball, this process is called hydrodynamic separation. The extreme compression of the atmosphere, by the blast wave, causes the air in front of the fireball to be heated to incandescence. Hence, after the hydrodynamic separation the fireball actually consists of two concentric regions:

- 1). Hot inner core known as the isothermal sphere
- 2). Outer layer of luminous shock-heated air.

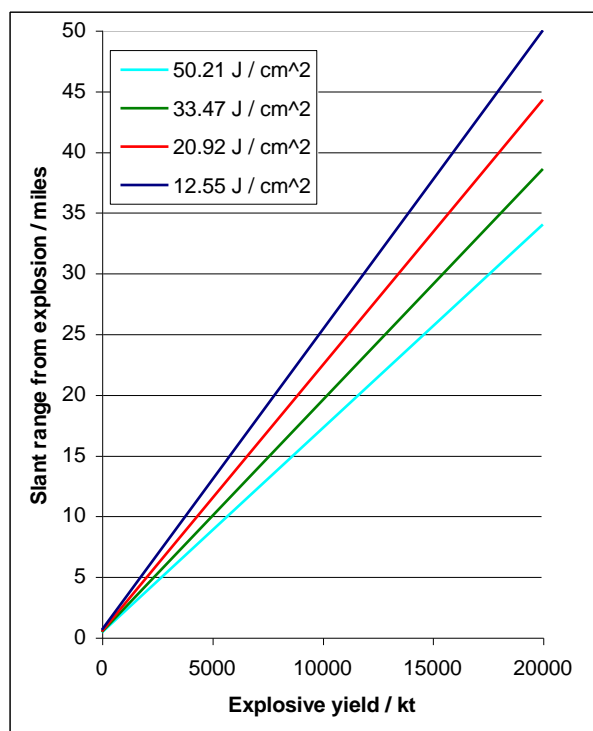
The outer layer initially absorbs most of the radiation from the isothermal sphere causing the apparent surface temperature of the fireball and the intensity of radiation emitted from it to decrease after separation. As the shockwave front advances further, the temperature of the shocked air diminishes and becomes increasingly transparent. This results in the unmasking of the incandescent isothermal region and increases the apparent surface temperature of the fireball. This phenomenon is referred to as "breakaway".

Rate of thermal emission

The rate of thermal emission from the fireball depends on its surface temperature. The thermal output of the nuclear air burst will then occur in two pulses: an initial phase consisting mainly of ultra violet radiation which contains 1% of the

total radiant energy of the explosion and is terminated as the shock front supersedes the fireball, the second pulse occurs after breakaway, Figure 9 illustrates the two pulses.

It is the second thermal pulse from the fireball that is responsible for most of the thermal effects. Thermal exposure, measured in joules per meter squared of the exposed surface, will be less further from the center of the explosion because the radiation is spread over a greater area and is attenuated in passing through the intervening air. The quantity



of thermal radiation at any given point varies approximately with the square of the distance from the explosion; this is because the fireball is close to the point source thermal radiation. The inverse square relationship does not hold because thermal radiation, especially UV radiation, will be absorbed and scattered by the atmosphere. Atmospheric visibility affects the attenuation of thermal energy with distance to a limited degree; the decrease in transmission is largely compensated by an increase in scattered radiation.

2.1.3.5 Yield and altitude

The yield of a nuclear explosion is directly proportional to the total amount of thermal radiation, period of time, which it is emitted, and the range of the thermal effects as shown in Figure 10.

Altitude effects: The thermal radiation intensity as a function of position will depend in the altitude and type of burst. Low altitude bursts cause the most thermal hazard. The general thermal effects will be less for surface bursts, and non for subsurface bursts. During surface bursts a large part of the energy is absorbed by the ground and water around ground zero. Natural shielding due to the environmental terrain

obstacles like dust, moisture and various gases in the air near the surface of the earth will reduce the thermal energy reaching the target. In subsurface bursts of low yields, most of the thermal energy is absorbed and dissipated in heating and vaporizing soil and water below the surface.

High altitude effects (> 30km above the surface of the earth): the low density of the atmosphere changes the nature of the thermal radiation process because the primary thermal radiation is absorbed in a much larger volume of air, which

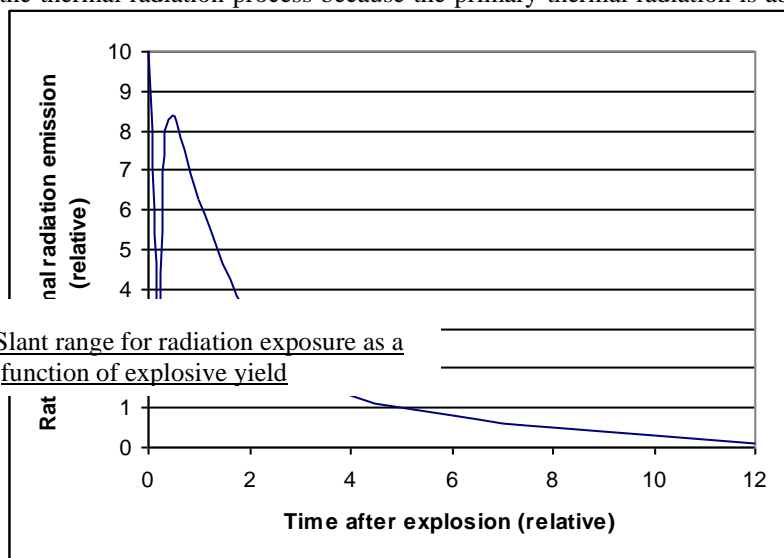


Figure 10 Slant range for radiation exposure as a function of explosive yield

implies that the overall temperature of given out to the atmosphere is far less than the low altitude bursts. Much of the radiation is emitted so slowly that it is ineffective. Thermal radiation accounts for a large percentage of the explosive yield. Approximately 25-35% of the total yield is emitted in a single pulse burst. The intensity of radiation on ground level is low due to the distance between the center of the burst and the surface of the earth below.

2.1.4 Assumptions

The scope of the physics model that will be integrated into the program will be defined in this section. During the detonation the energy given out by the weapon is divided into three parts which give rise to separate effects:

- 1). Blast Wave
- 2). Thermal radiation
- 3). Nuclear radiation

Figure 9 Two-pulse thermal emission during an airburst

Blast wave will be neglected: the explosion causes hot gaseous residues to move spherically outwards from the point of the blast, carrying with it a thin shell of residual material, this shell is called the hydrodynamic front. This causes the surrounding medium to compress due to the shockwave, this shockwave causes sheering effects on the obstacles in its path due to abrupt changes in atmospheric pressure. Modelling these mechanical effects is outside the scope of this project, and hence will be neglected.

Nuclear radiation will be neglected: this consists of two types of radiation: particle emission (alpha / beta) and radiation (gamma). The effects of these two types of radiation can be found during the explosion (initial radiation) and some long period after the explosion (residual radiation).

Initial radiation is ionizing radiation emitted within the first minute post detonation as its source is almost entirely from the nuclear processes occurring at the detonation. Approximately 5% of the explosive yield released by the nuclear airburst is transmitted into initial neutron and gamma radiation. The effects of the radiation do not effect the internal temperature of the vessel that we are modelling, the only possible effect of the nuclear radiation upon the target would be causing secondary radiation to occur across the material of the vessel, but to model the nuclear radiation effects upon the vessel is external to the scope of the program. The same applies to residual radiation: it has no effect on the internal temperature of the vessel. More importantly, the vessel will be unmanned, which implies the vessel remains unaffected because it contains no human personnel on-board that would be exposed to radiation hazards. Nuclear shielding is suffice to protect the computer instrumentation on board. Moreover, the range for significant levels of initial radiation does not increase with weapon yield which implies the initial radiation becomes less of a hazard with increasing yield, with weapons above 50kt, blast and thermal effects are much greater in importance whereby the radiation effects can be ignored.

Residual radiation is the radiation, which is emitted later than one minute after detonation and occurs due to the decay of the radioisotopes produced during the explosion. They are produced by:

- 1). Fission products: produced when heavy uranium or plutonium nuclear is split in a fission reaction. They emit beta and gamma radiation.
- 2). Unfissioned nuclear material: due to the inefficiency of the nuclear weapon, not all the material is fissioned; much of the uranium and plutonium are blown apart from their assembly. These unfissioned materials undergo decay via the emission of alpha particles, which are insignificant with respect to gamma and beta radiation.
- 3). Neutron-induced activity: Any atomic nucleus exposed to a neutron flux will capture neutrons and become radioactive then undergo decay via emission of beta and gamma radiation over a long period of time. The neutrons emitted, as part of the initial nuclear radiation, will cause secondary activation of weapon residues. More importantly, the environmental material (the surrounding medium): soil, air, and water will be activated depending on their composition and distance from the point of detonation. The target vessel, near ground zero, may be made of metals, which may become activated by neutrons given off from the initial radiation or secondary emission from activated materials within the soil such as sodium, manganese, aluminium or silicon, but these are negligible hazards because of the limited area involved as well as the low intensity.

Surface burst model: for this experiment, surface bursts will only be modelled; this is to avoid the complications of nuclear fallout involved in airbursts.

Fireball profile: the fireball is initially spherical, this can be approximated to a rectangle in two dimensions, and this is much easier to model, as we do not need accuracy in modelling the fireball physics to high accuracy. It will be seen later that this approximation makes it easier to implement than a spherical profile.

Energy output of fireball: referring to figure 9, it shows that the fireball has two radiation emission peaks, this characteristic is due to the shock front from the burst overtaking the fireball. We shall only be concerned with the second thermal energy peak, and the decline in thermal energy emission that follows.

Environmental temperature: the environmental temperature and pressure will be initialised at 297 degrees K (24 degrees centigrade).

2.2 The Heat Shielding

2.2.1 Engineering requirements

Radiation exposure, measured in Joules/cm², increases with the yield of the weapon. For weapons of lower yield, the thermal pulse is short, meaning that the exposed surfaces cannot cool quickly enough and undergo thermal damage. The requirements needed to be fore filled by the heat shielding is as follows:

Good thermal radiation reflector: Incident radiation from the fireball will be partly reflected and absorbed by the shielding panels. The radiation that is absorbed depends on the material and its colour: A thin material may transmit a large percentage of the radiant energy striking it whereas a light coloured material may reflect much of the incident radiation hence eluding damage thermal damage. The absorbed thermal radiation raises the temperature of the absorbing surface resulting in scorching, charring or ignition of combustible organic materials within the vessel it is protecting.

Good thermal insulator (low thermal conductivity, k): This confines the absorbed thermal energy to a superficial layer of the material. In other words, the thermal conductivity defines the rate at which heat passes through small area, A,

inside the body and is given by: $Q = kA \frac{dT}{dx}$; Where $\frac{dT}{dx}$ is the temperature gradient normal to the area A in the

direction of the heat flow, and k is the thermal conductivity of the body at temperature T. The units of thermal conductivity, k, are: J s⁻¹ m⁻¹ K⁻¹. The materials chosen for heat shielding should have a thermal conductivity lying between a lower limit of 0.05 J s⁻¹ m⁻¹ K⁻¹ and an upper limit of 0.8 J s⁻¹ m⁻¹ K⁻¹.

High specific heat capacity, c: The heat shielding should have the ability to withstand high level of thermal radiation exposure without the danger of melt down. The heat shielding will be exposed to the thermal E-M radiation emitted directly from the fireball as well as the high temperatures of the atmosphere; this means that a large quantity of heat will be absorbed via conduction to the heat shielding panels. It is important, therefore, that the heat shielding act as a good heat sink to the high operating temperatures. The heat shielding should not react abruptly to the changes in external temperature, thus the material should have a high specific heat capacity: this value represents the quantity of heat required to raise the temperature of one kilogram of a substance by one Kelvin. Its units are: J kg⁻¹ K⁻¹. The heat shielding should have a specific heat capacity lying between a lower limit of 1500 : J kg⁻¹ K⁻¹ and an upper limit of 2400 J kg⁻¹ K⁻¹.

High density, ρ: The material will undergo some degree of sheering and flexing should it be hit by a projectile or go against the shock front of the blast. The high density of the heat shielding is necessary to contribute to the vessel's inertial stability. The heat shielding is required to have a lower limit of 1000 kg m⁻³ to 2000 kg m⁻³

2.2.2 Materials considered

Listed below are the candidate materials that comply with the engineering criteria outlined above. These materials will be implemented into the system database, so the user can easily select the appropriate material that would best perform under simulated thermal conditions.

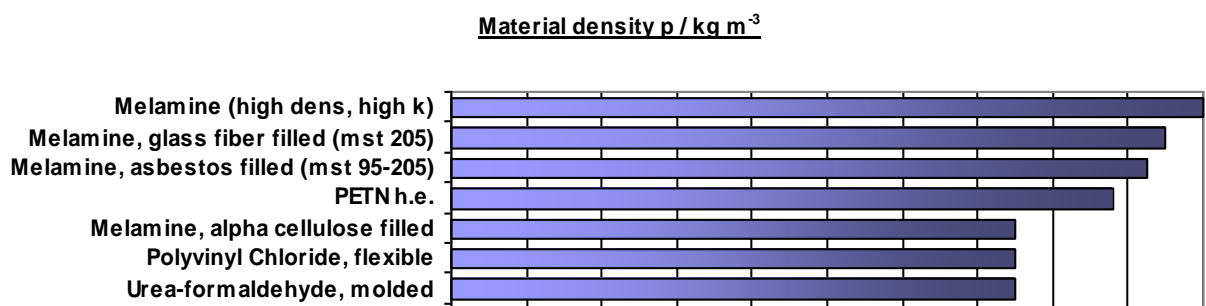
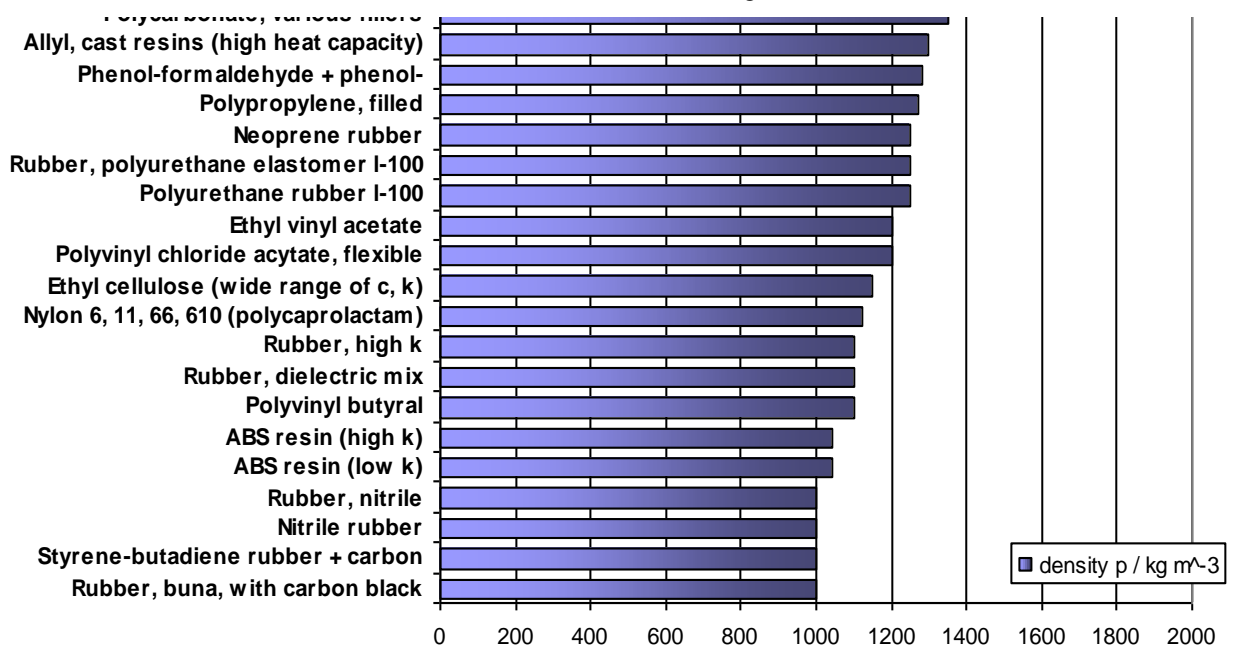


Table 3 Densities of heat shielding materials



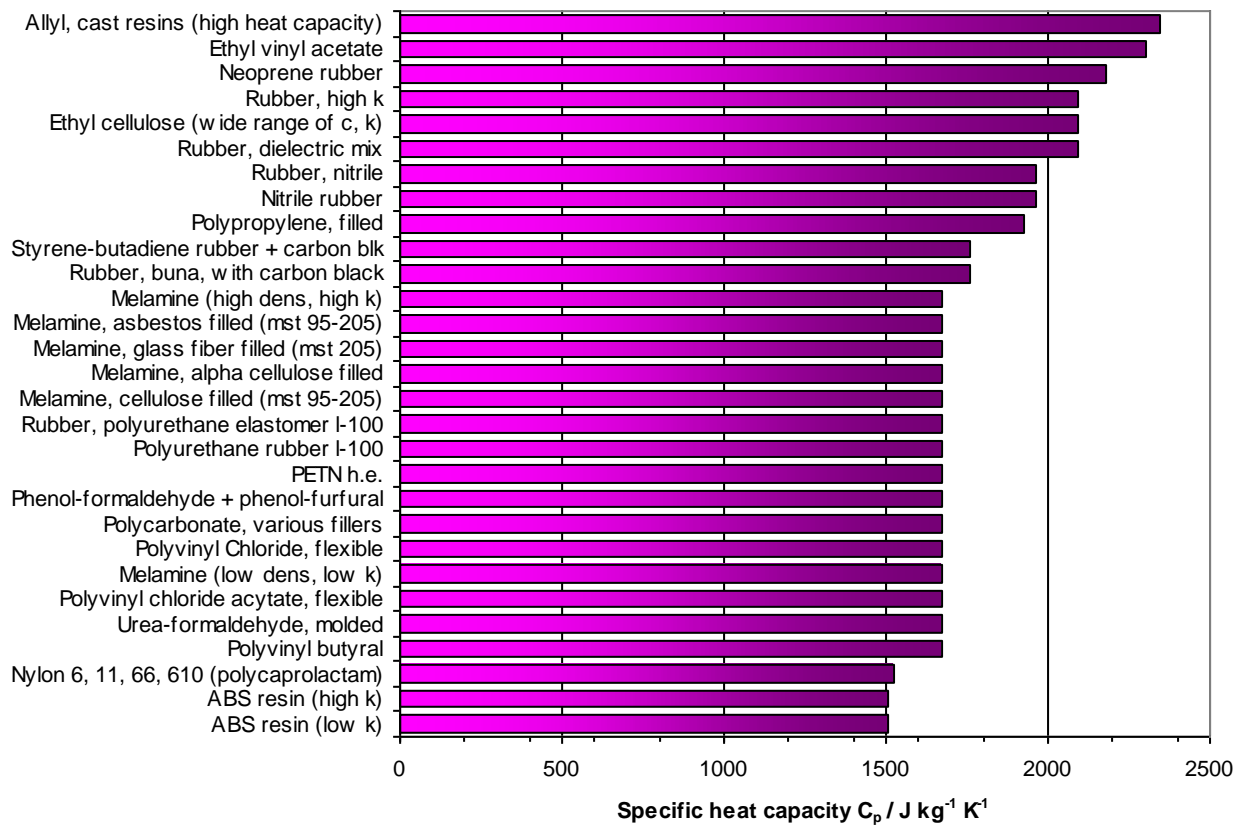


Table 4 Specific heat capacity of heat shielding materials

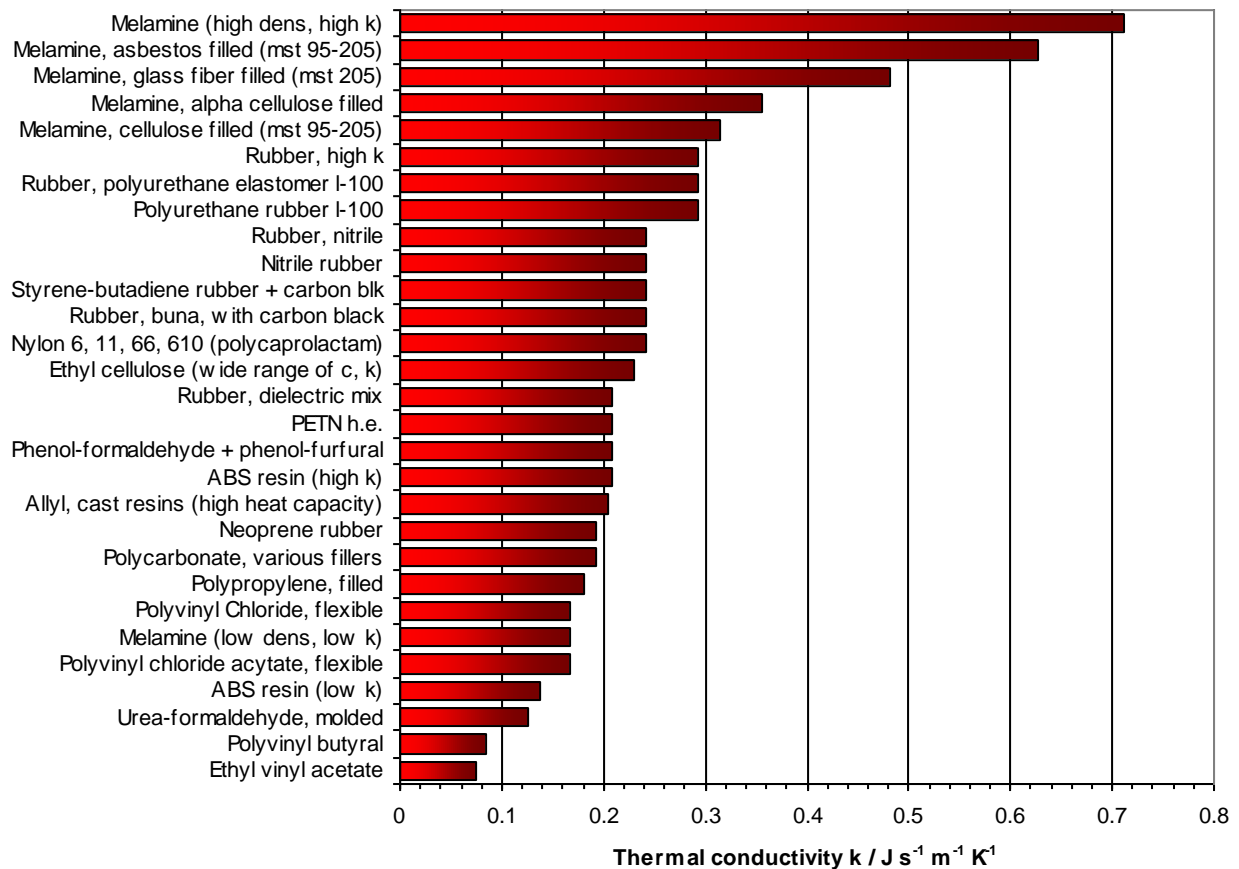


Table 5 Thermal conductivity of heat shielding material

2.2.3 Assumptions

The following assumptions will be made when developing the computer program:

- 1). The operating temperature inside the vessel should not exceed $393 \text{ K} \pm 10 \text{ K}$. The program should be able to calculate the optimum thickness of the specified material to meet this requirement. This temperature is the maximum operating temperature of the computer equipment that would be on board the unmanned vessel, it depends upon the heat shielding for protection against thermal damage.
- 2). The structural integrity of the heat shielding materials will not be evaluated; it is external to the scope of the program design criteria.
- 3). It can be assumed that the heat shielding will be divided into small square panels of surface area 225 cm^2 , i.e. the surface geometry remains constant in thickness and dimension. According to the design parameters set by the vessel engineers, the heat shielding panels are only allowed to have a maximum thickness of 10 inches.
- 4). The candidate materials might be good insulators, but they have extremely poor optical reflectivity. These materials do not have the same reflectivity as metals, which can reflect radiation much better. So it may be assumed that the heat shielding panels be encased in a steel envelope that would act as a thermal radiation reflector shield as well as a protection from corrosive chemical substances. This will make the heat shielding panels composite; this can be neglected since steel has a very high thermal conductivity –heat would pass through it via conduction without much ease. So the heat shielding material will be modelled as a single slab of homogenous material.

2.3 Solutions Procedure

In order to solve this problem effectively, we have to examine the events, which occur during detonation, and assess the necessary variables needed to input into the thermodynamics equations. The computerisations of these equations into algorithms ready to be implemented into the system are discussed in this section.

2.3.1 Numerical treatment

The events that occur during a nuclear detonation are discussed above, from which a model of the problem can be fleshed out. The problem is broken up into three stages;

Stage 1: Obtaining the temperatures contained around the fireball: here we take the thermal energy component from the warhead yield and use the algorithm outlined in section 2.3.2 to obtain a matrix representing the thermal source. The temperatures obtained from this matrix are passed to the second stage to be used as initial boundary values.

Stage 2: Using the boundary value temperatures from stage 1, the temperature distribution across the rest of the environment will be calculated using the one-dimensional steady state formulae shown in Equation 1.

$$\frac{d}{dx} \left(kA \frac{dT}{dx} \right) - hP(T - T_{\infty}) = 0$$

Equation 1 One dimensional steady state diffusion equation

The formula given in equation 1 is in one dimension along the x-axis. This formula will be adapted to be used to solve for all points in the 2d plane. Details of the meaning of the terms and the discretisation of this equation are given in section 2.3.3.

Stage 3: After obtaining a matrix of all the temperatures in the locality of the bomb, the temperature of the cell containing the vessel with the heat shielding is found and applied to equation 2 given below.

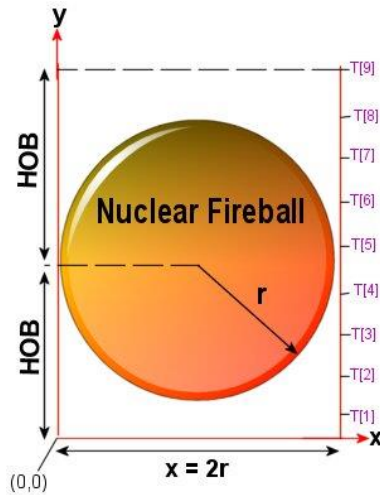
$$Q = kA \frac{dT}{dx}$$

Equation 2 Fourier heat conduction equation

The internal temperature of the vessel can then be determined for varying values of thermal conductivity, k , and hence different materials. The success or failure of the material will depend on whether the internal operating temperature of the vessel exceeds the limit of $393\text{ K} \pm 10\text{ K}$. The computer program will have additional functionality of finding the best-suited material for the specified conditions; reading in all the materials from the database and calculating the minimum thickness of material needed to achieve the desired results. The algorithmic adaptation of this method is outlined in section 2.3.4.

The databases of values needed to perform these calculations have been derived and developed using information from this section. The algorithms given in this section are written in Java and have been directly implemented into the program with modification to the interface modules.

2.3.2 Algorithm for determining the boundary value temperatures



The temperatures generated from the nuclear fireball along each point on the line $x=2r$ need to be obtained, where $2r$ is the diameter of the fireball. Figure 11 represents the fireball approximation, and the region which will be modelled in order to calculate the boundary temperatures $T[n]$. These boundary temperatures will be used in Stage two of the calculation to determine an environmental thermal image. At this point it is convenient to define the database definitions of the table “WARHEAD” (Table 6); as the algorithms in this chapter will use them. The method to develop the algorithm in Java is outlined in this section.

Post-processing

Step 1: Find the temperature of the fireball. This is obtained from the yield of the warhead multiplied by the efficiency. The yield, which represents the total amount of energy given out by the weapon due to the nuclear reaction that occur at the point of detonation. Referring back to the chart in Figure 10, it shows that the yield, total output energy, is broken up into four different forms of energy, hence we can see the percentage of energy distribution.

Figure 11 Fireball boundary temperatures in a 2d plane

Field	Type	Description
nukeID	Integer	Unique database key for each weapon
nukeName	String	Name of nuclear weapon
nukeType	String	Gun Assembly / Implosion assembly
nukeYield	Real	Total energy output of the weapon Joules/sec
HOB	Real	Height of blast / meters: this is the distance from the centre of the spherical fireball to the surface of the earth.
fireballRadius	Real	The maximum radius that the fireball can grow to. / meters
efficiency	Real	Value between 0.00 to 1.00. This is the percentage of nuclear material that undergoes fission/fusion during the explosion. It directly affects the yield of the weapon, i.e. the total energy output. Default value = 1.00 (i.e. 100% efficient).

Table 6 Database table of "WARHEAD"

Multiplying the yield by the efficiency and then taking 20%, gives us the thermal portion of the yield. Approximating the fireball to a perfect sphere, and assuming that the energy is distributed evenly across every point in the sphere, the energy density can be obtained using the formula given in Equation 3 below.

$$U_{\text{internal}} = m_{\text{air}} C_p (T_{\text{fireball}} - T_{\text{ambient}})$$

Equation 3 Internal thermal energy of the fireball

Where;

- U_{internal} – thermal energy component of the weapon’s yield (Joules/sec or Watts)
- m_{air} – average mass of air at normal atmospheric temperature and pressure (Kg)
- C_p – specific heat capacity of air, which will be taken as $1.00 \times 10^3 \text{ J kg}^{-1} \text{ K}^{-1}$
- T_{ambient} – ambient temperature of air prior to explosion (user specified) (K)
- T_{fireball} – average temperature of fireball (K)

Rearranging the formula to obtain T_{fireball} gives:

$$T_{\text{fireball}} = \frac{U_{\text{internal}}}{m_{\text{air}} C_p} + T_{\text{ambient}}$$

Equation 4

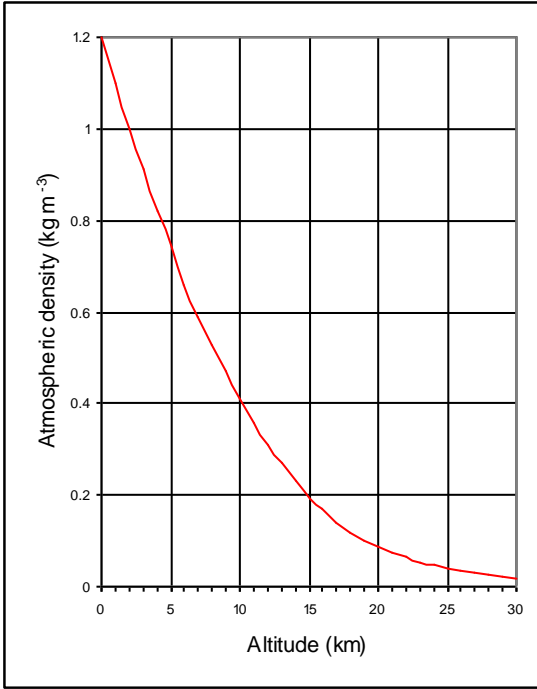


Figure 13 Graph representing the atmospheric density as a function of altitude

Because the density of air varies as a function of altitude, it can affect the mass of the fireball sphere and hence the fireball temperature. Therefore, the atmospheric density, and hence the mass of air, will be taken from a database table by specifying the HOB of the fireball. Figure 13 shows the variation of density as a function of altitude.

Taking into the consideration the spherical nature of the fireball, as well as the density variations the following expression for the mass of air can be substituted into equation 4:

$$m_{\text{air}} = \frac{4}{3} \rho_{\text{air}} \pi r^3 = \frac{4\pi r^3 \rho_{\text{air}}}{3}$$

Equation 5

Giving:

$$T_{\text{fireball}} = \frac{3U_{\text{internal}}}{4\pi r^3 \rho_{\text{air}} C_p} + T_{\text{ambient}}$$

Equation 6

The function in Figure 12 represents the Java implementation of the code. After attaining the average fireball temperature, a computer model needs to be designed to represent it. In order to achieve this have to create a numerical scheme which can be applied to the region shown in Figure 11. The process of bridging

over a physical model to a numerical one is called discretisation. The following steps show how the discrete model was created for the first stage of the program model. Referring to Figure 11, it shows that the region of interest is bounded

```
public double fireballTemperature(double uInternal, double fireballRadius, double airDensity,
double cPAir, double ambientTemp)
{
    tFireball =
    ((3.0)*(uInternal))/((4.0)*(Math.PI)*(Math.pow(fireballRadius,3))*(airDensity)*(cPAir)) +
    ambientTemp;
    return tFireball;
} // fireballTemperature
```

Figure 12 Java function to calculate the fireball temperature

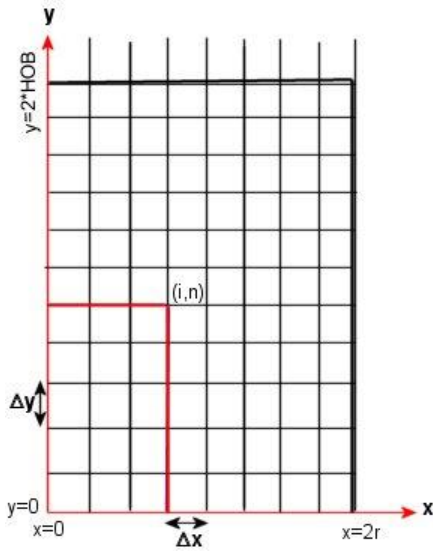
by the horizontal lines $y = 0$ and $y = 2*\text{HOB}$, and vertical lines $x = 0$ and $x = 2r$, note: HOB is the height of the blast and is $> r$ (radius of fireball). The purpose of modelling this area is so that the boundary temperatures may be calculated: $T[0]$, $T[1]$... $T[n]$.

Step 1

First, replace the independent variables, x and y , with discrete ones:

$$x_i = i\Delta x = \frac{i(2r)}{x_{\text{res}}} \quad i=0,1,\dots,x_{\text{res}}$$

$$y_j = j\Delta y = \frac{j(2*\text{HOB})}{y_{\text{res}}} \quad j=0,1,\dots,y_{\text{res}}$$



Both x_i and y_i are closed domain. A 2-D grid is generated over the region of interest, as shown in Figure 14, where $\Delta x = 2r/x_res$ and $\Delta y = 2*HOB/y_res$. A two dimensional matrix of size $A[x_res][y_res]$ is required to hold all the temperatures at each node represented by the meshed region in Figure 14.

Step 2

The coordinates representing the centre of the fireball sphere have to be discretised. Let it be assumed that the centre of the fireball lies at xy coordinates (a,b) in the normal x-y plane, where $a = r$ and $b = HOB$. Thus, let i_a and j_b represent the coordinates of the centre of the fireball, then:

$$i_a \Delta x = r \Rightarrow i_a = r / \Delta x$$

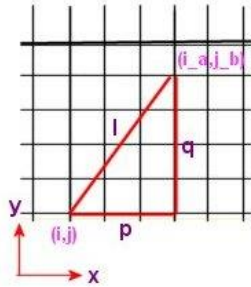
$$j_b \Delta y = HOB \Rightarrow j_b = HOB / \Delta y$$

Figure 14 Discretisation (meshing) of fireball region

so the centre of the fireball lies on the discretised coordinates (i_a, j_b) which is then $(r/\Delta x, HOB/\Delta y)$. Note, when calculating values i_a, j_b we round up to the nearest integer.

Step 3

Identify all the points, which lie on the fireball in the discretised plane and assign the fireball temperature to them. This is done, simply by checking each point (node) on the grid to see if it's distance from the centre (i_a, j_b) of the fireball is greater or less than the radius of the fireball, if it is, then we assign it the fireball temperature to that node. If the node does not lie on the fireball, then the ambient atmospheric temperature is assigned. Figure 15 illustrates this method.



$$p = |i_a \Delta x - i \Delta x|$$

$$q = |j_b \Delta y - j \Delta y|$$

$$l = \sqrt{p^2 + q^2}$$

Figure 15 Length between two points

The length, l , is calculated for each point on the node, then tested to see if it satisfies the condition that $l \leq r$, i.e. to see if the point lies on the fireball. The java algorithm that performs this process is shown in Figure 16.

```
public void initFireballMatrix(double fireballTemp, double fireballRadius,
    double ambientTemp, double HOB, int x_res,
    int y_res, double a[][])
{
    double deltaX = (2*fireballRadius)/x_res;
    double deltaY = (2*HOB)/y_res;
    double p, q, length = 0.0;

    // set centre of fireball coordinates using new discretised grid value (a,b)

    int i_a = (int)(fireballRadius/deltaX);
    int j_b = (int)((HOB)/deltaY);
    // cont.
```

```

// cont.
// We now recurse through the array a[][] and assign the relevant temperatures
// to the nodes on the mesh.

for (int j = 0; j < y_res; j++)
{
    for(int i = 0; i < x_res; i++)
    {
        p = Math.abs(i_a*deltaX - i*deltaX);
        q = Math.abs(j_b*deltaY - j*deltaY);
        length = Math.sqrt(p*p + q*q);

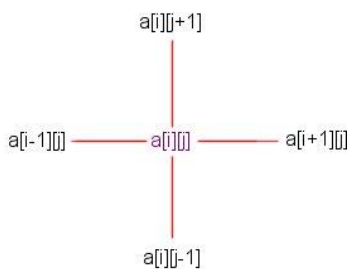
        // test to see if the point lies on the fireball or out it.
        // and assign the relevant temperature accordingly
        if (length <= fireballRadius) {a[i][j] = fireballTemp;}
        else {a[i][j] = ambientTemp;}
    } // nested for loop
} // outer for loop

} // initFireballMatrix

```

Figure 16 Java algorithm to initialise set the temperature at nodes

Step 4



Now that the temperature values have been assigned to all the node points in the matrix $a[][]$, an averaging filter must be applied to it in order that the fireball temperatures diffuse with the ambient temperatures, making the thermal model around the fireball more realistic. The method: every point in $a[][]$ is averaged out with it's neighbours to the North, South, East and West of it's position, as shown in Figure 17, where the average is given by:

$$a[i][j] = (1/4) * (a[i+1][j] + a[i-1][j] + a[i][j+1] + a[i][j-1]).$$

Figure 17 Averaging method

The new averaged value is placed in a new temporary matrix $aTemp[][]$. After one pass, all the values in $aTemp$ are copied back to $a[][]$ and the process is repeated n times (specified by the user).

Because values that lie on the border of the matrix $a[][]$, it is not possible to obtain all the values for averaging. For example, at the point $a[0][0]$ we cannot obtain the values $a[-1][0]$, and $a[0][-1]$. So, to solve this problem a dummy border is set up around the matrix $a[][]$, and dummy values assigned to this matrix is the ambient temperature of the atmosphere. But the matrix $a[][]$ will not be expanded to accommodate this border, instead the temporary matrix $aTemp[][]$ will have the border, and the values from $a[][]$ will be copied over to it. Thus the size of $aTemp$ will be $aTemp[x_res+2][y_res+2]$. The java implementation of this method is given in Figure 18.

```

public void createTempMatrix(double a[], double aTemp[], double ambientTemp,
                             int x_res, int y_res)
{
    // Create dummy border temperatures
    // ... top and bottom rows
    for (int i = 0; i < (x_res + 2); i++)
    {
        aTemp[i][0] = ambientTemp;
        aTemp[i][y_res + 1] = ambientTemp;
    } // cont.
}

```

```

// ... left and right columns
for (int j = 0; j < (y_res + 2); j++)
{
    aTemp[0][j] = ambientTemp;
    aTemp[x_res + 1][j] = ambientTemp;
}

// copy all the temperature values over from matrix a[][] into aTemp[][]
for (int j = 0; j < (y_res); j++)
{
    for (int i = 0; i < (x_res); i++)
    {
        aTemp[i + 1][j + 1] = a[i][j];
    }
}

} // createTempMatrix

```

Figure 18 Java method to create a the dummy border and initialise temporary matrix

Now that the temporary matrix is setup, it has to passed through the averaging filter algorithm. The new averaged values will be placed back into the original matrix, `a[][]`. The purpose of averaging the temperatures around the fireball is to simulate heat diffusion; the fireball as the source and the immediate system around the fireball being the atmosphere at ambient temperature. It is assumed that the fireball has grown to it's maximum size and is modelled approximately ten seconds after the point of detonation, and the temperatures in matrix `a[][]` (post-averaging) represent the thermal image of the region around the fireball. The algorithm in Figure 19 represents the averaging filter in java.

```

public void averagingFilter(double aTemp[][], double a[][], int x_res, int y_res, int recursion)
{
    /* We run the averaging filter over aTemp[][] placing the new values into the matrix a[], hence
    overwriting the previous values of a[] */
    int counter = 1;

    while (counter < recursion + 1)
    {
        /* The region that will be averaged in aTemp[][] is covered by aTemp[1][1] to a[x_res][y_res] */

        for (int j = 1; j < (y_res + 1); j++)
        {
            for (int i = 1; i < (x_res + 1); i++)
            {
                a[i-1][j-1] = (0.25)*(aTemp[i-1][j] + aTemp[i+1][j] + aTemp[i][j+1] + aTemp[i][j-1]);
            } // nested
        } // outer

        /* We need to keep a copy of the new matrix a[][] back into the aTemp[][]. We need to do this in order to
        repeat the process should the recursive value be greater than 1 */

        matrixCopy(a, aTemp, x_res, y_res);
        counter = counter + 1;
    } // while loop
} // averagingFilter

```

Figure 19 Averaging filter algorithm

Once the temperatures have been averaged in the fireball region, the results of $a[][]$ are sent to a display function to be output to the user console in graphical format. The routine for displaying the temperatures will be shown in the next chapter. The boundary temperatures shown in Figure 11 need to be obtained from the matrix $a[][]$ in order to begin the processing of Stage 2. The required temperatures are held in the array elements :

$$a[x_res-1][0], a[x_res-1][1], a[x_res-1][2], \dots, a[x_res-1][y_res-1]$$

: these values will be copied over to a column matrix $bTemp[]$.

2.3.3 Algorithm for convective heat transfer in 2d

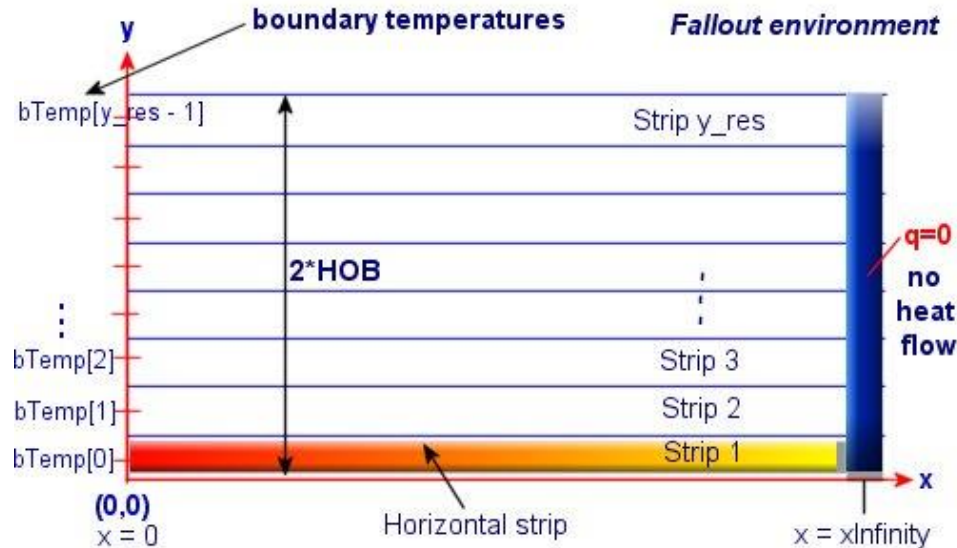


Figure 20 Fallout environment model

The diagram in Figure 20 represents the environmental region immediately to the right of the fireball region (not shown). The environment spans a range of $x = 0$ to $x = xInfinity$ to $y = 0$ to $y = 2*HOB$. Further, the region is divided into y_res horizontal strips. Convective heat transfer occurs along the length of each strip of air, with the fireball being the heat source at point $x = 0$ (not to be confused with the fireball region $x = 0$), and $x = xInfinity$ being the heat sink, i.e. where the thermal energy from the fireball is *almost* fully absorbed into the atmosphere. Thus heat from the fireball is convected along the strip up to the point $x = xInfinity$ where thermal energy is minimal. The distance $xInfinity$ varies for different yields from nuclear weapons. Table 7 represents distance from the blast, $xInfinity$ (more precisely the slant range from mid-point of fireball to the surface $xInfinity$ away where the thermal effects are minimal) and the corresponding explosive yield (in kT) of the warhead. These values are based on experimental data.

Explosive Yield (kT)	Slant range from point of detonation: $xInfinity$ / km (Energy flux 3 cal / sq cm)	Slant range from point of detonation / miles
1	0.965	0.60
5	1.770	1.10
10	2.012	1.25
50	2.253	1.40
100	8.047	5.00
500	16.093	10.00
1000	32.187	20.00
5000	64.374	40.00
10000	77.249	48.00
20000	80.467	50.00
> 20000	96.561	60.00

Table 7 Explosive yield and corresponding minimal thermal effects range

Table 7 will be implemented into the database design in order that the computer program may select the appropriate value for $xInfinity$ by specifying the value for explosive yield.

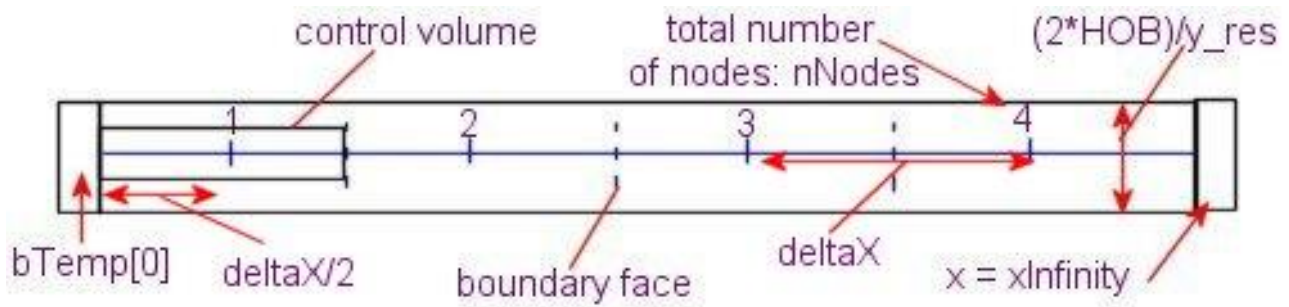


Figure 21 Two-dimensional horizontal strip of air

Referring to Figure 21:

nNodes : total number of nodes in one strip (user specified)

deltaX – distance between nodes in meters, $\text{deltaX} = (\text{xInfinity}/\text{nNodes})$

Now let us examine each horizontal strip individually, referring to Figure 21 above. The strip can be thought of as a rectangular block of air in three dimensions, with uniform cross-sectional area A. Each strip has a height of $(2*HOB)/y_res$. It will be assumed that the width of the block is $2r$ (twice the radius of the fireball), thus the cross sectional area A, is given by height x width:

$$A = \frac{(2*HOB)*(2r)}{y_res} = \frac{4r*HOB}{y_res}$$

Figure 22 Cross sectional area of strip

Before the detonation, each strip is in thermal equilibrium with the ambient temperature of the surrounding atmosphere. We will ignore the heat loss/gain effects from adjacent strips. Every strip, y_res in all, will be modelled in one dimension. Thus the Equation 1 governs the one-dimensional heat transfer across this strip, namely:

$$\frac{d}{dx} \left(kA \frac{dT}{dx} \right) - hP(T - T_\infty) = 0$$

Where:

h (constant) – convective heat transfer coefficient of air = $30 \text{ W m}^{-2} \text{ K}^{-1}$ (at 300K)

P (constant) – perimeter of strip in meters = $2*(\text{xInfinity} + 2*HOB/y_res)$

k (constant) – thermal conductivity of air taken to be $0.025 \text{ W m}^{-1} \text{ K}^{-1}$ (at 293K)

T_∞ - ambient atmospheric temperature / K (= ambientTemp in code)

$-hP(T - T_\infty)$: convective heat loss, sink term

The analytical solution to equation 1 is:

$$\frac{T - T_\infty}{T_B - T_\infty} = \frac{\cosh[n(L - x)]}{\cosh(nL)}$$

Equation 7 Analytical solution to governing equation

where:

$$n^2 = \frac{hP}{kA}$$

and, $L = \text{length of the strip} = \text{xInfinity}$. Figure 23 represents the Java implementation for determining n^2 .

```
public double nSquared(double hCoeff, double HOB, int y_res, double xInfinity,
    double kAir, double fireballRadius)
{
    double crossSectionA = 0.0;
    double perimeter = 0.0;
    double nSqr = 0.0;

    // calculate the cross sectional area, A, of the strip
    crossSectionA = ((4.0)*(fireballRadius)*(HOB))/(y_res); // cont.
```

```

// ... cont.
// calculate perimeter P of the strip
perimeter = (2.0)*(xInfinity + ((2.0)*(HOB))/(y_res));

// calculate n squared
nSqr = ((hCoeff)*(perimeter))/((kAir)*(crossSectionA));
return nSqr;

} // nSquared

```

Figure 23 Java implementation to calculate n^2 value

Numerical solution of governing equation using finite volume method

Step1

Divide the strip into a grid of n number of nodes ($n = nNode$ in code - see Figure 21). Thus the strip is divided into $nNode$ control volumes. Rewriting the equation 1 as:

$$\frac{d}{dx} \left(\frac{dT}{dx} \right) - n^2 (T - T_\infty) = 0$$

Equation 8

Integrating equation 8 over the control volume gives:

$$\int_{\Delta V} \frac{d}{dx} \left(\frac{dT}{dx} \right) dV - \int_{\Delta V} n^2 (T - T_\infty) dV = 0$$

Equation 9

$$\left[\left(A \frac{dT}{dx} \right)_e - \left(A \frac{dT}{dx} \right)_w \right] - [n^2 (T_p - T_\infty) A \delta x] = 0$$

Equation 10

- Where subscript e and w represent the East and West boundaries of the control volume. T_p represents the temperature of the current node. A formula for the interior nodal points, represented by nodes 2 and 3 in our sample strip in Figure 21, is obtained by replacing the differential terms in Equation 10 using linear approximations, thus

$$\left[\left(\frac{T_E - T_p}{\delta x} \right) - \left(\frac{T_p - T_w}{\delta x} \right) \right] - [n^2 (T_p - T_\infty) \delta x] = 0$$

Equation 11 Linear approximation of solution for interior nodes

Where T_E and T_w represent the East and West boundary temperatures to T_p respectively. Rearranging Equation 11 gives:

$$\left(\frac{1}{\delta x} + \frac{1}{\delta x} \right) T_p = \left(\frac{1}{\delta x} \right) T_w + \left(\frac{1}{\delta x} \right) T_E + n^2 \delta x T_\infty - n^2 \delta x T_p$$

Equation 12 Linear form of solution

Thus, for the interior nodal points a general linear formula can be written as:

$$a_p T_p = a_w T_w + a_e T_E + S_U$$

Equation 13 General linear equation for interior nodes

S_U is the source term, and the coefficients of Equation 13 are given in Table 8.

a_w (aWest)	a_E (aEast)	a_P (aPoint)	S_P (sPoint)	S_U (sU)
1/deltaX	1/deltaX	aWest + aEast - sPoint	-(nSqr)*(deltaX)	nSqr*deltaX*ambientTemp

Table 8 Coefficients for interior nodal points

(Note: $n^2 = nSqr$. $\delta x = \text{deltaX}$. $T_\infty = \text{ambientTemp}$.)

Next, we apply the boundary conditions at the East and West nodes (two end nodes), points 1 and 4 in the example in Figure 21.

At node 1 the West control volume boundary is kept at a temperature of $bTemp[0]$, so Equation 9 will be integrated over this range, giving :

$$\left[\left(\frac{T_E - T_P}{\delta x} \right) - \left(\frac{T_P - T_B}{\delta x/2} \right) \right] - [n^2 (T_P - T_\infty) \delta x] = 0$$

Equation 14 Linear solution for west node point

So the coefficients of the discretised equation at boundary node 1 are given in Table 9.

a_w (aWest)	a_E (aEast)	a_P (aPoint)	S_P (sPoint)	S_U (sU)
0	1/deltaX	aWest + aEast - sPoint	-(nSqr)*(deltaX)	nSqr*deltaX*ambientTemp

Table 9 Coefficients for west boundary node point

At the last node, node $nNode$ (node 4 in Figure 21), the heat flux across the east side boundary of the control volume is approximately zero, since it is the point where the thermal effects of the fireball are negligible, thus integrating Equation 9 over this volume gives:

$$\left[0 - \left(\frac{T_P - T_W}{\delta x} \right) \right] - [n^2 (T_P - T_\infty) \delta x] = 0$$

Equation 15 Linear solution for east node point

Hence the east side coefficient is zero. There are no additional source terms associated with the zero flux boundary conditions. The coefficients at the east side boundary node ($nNode$) are listed in Table 10.

a_w (aWest)	a_E (aEast)	a_P (aPoint)	S_P (sPoint)	S_U (sU)
1/deltaX	0	aWest - sPoint	-(nSqr)*(deltaX)	nSqr*deltaX*ambientTemp

Now that we have a set of linear equations that need to be solved for temperature for all nodes on the strip, a matrix can be compiled of the coefficients, thus for four node points the matrix would look like that in Figure 24.

$$\begin{pmatrix} {}^1a_p & {}^1a_E & 0 & 0 \\ {}^2a_w & {}^2a_p & {}^2a_E & 0 \\ 0 & {}^3a_w & {}^3a_p & {}^3a_E \\ 0 & 0 & {}^4a_w & {}^4a_p \end{pmatrix} \begin{pmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}$$

Figure 24 Matrix system of linear equations

This matrix is called a tri-diagonal matrix, because it has a diagonal band of non-zero values and the other elements are zero. There is a special algorithm used to solve this type of matrix, it is known as Crout's algorithm. In order to solve this matrix, a Java algorithm has to be designed, where the matrix $a[][]$ represents the coefficient matrix and $b[]$ the column matrix in Figure 24. These two matrices are passed to the Crout's algorithm where a series of steps outlined in the code has to be performed. The matrix $a[][]$ needs to be decomposed into two parts: Upper triangular matrix ($U[][]$) and the lower triangular matrix ($L[][]$). The java implementation of this algorithm is given in Figure 25. The array $x[]$ contains all the temperatures that lie on the nodes along the strip. This process is repeated for all the strips and the temperatures are placed in an array called $environmentTemp[][]$ which represents the overall temperature profile in the environment due to the nuclear detonation. The matrix $environmentTemp[][]$ will be sent to a display class to be displayed graphically on screen, this will be explained in the next chapter. The important part of this calculation is

obtaining the temperature values that lie along the strip where the vessel heat shielding is placed. This means passing on the array x[] for the first strip in the calculation on to Stage 3 of the model.

```
public void croutSolver(double l[][], double u[][], double a[][], double b[], double z[], double x[], int nNode)
{
    /* This routine solves ax=b by means of decomposition into it's lower (l[][])
    and upper (u[][]) matrices. Final solution is stored in x[] */

    // Step 1 set l_1,1 = a_1,1 where notation is l_{row,column}
    l[0][0] = a[0][0];

    // set u_1,2 = a_1,2/l_1,1
    u[1][0] = a[1][0] / l[0][0];

    // Step 2: for i=2,...n-1
    for (int j = 1; j < nNode-1; j++)
    {
        // l_i,i-1 = a_i,i-1
        l[j-1][j] = a[j-1][j];

        // l_i,i = a_i,i - l_i,i-1 * u_i-1,i
        l[j][j] = a[j][j] - (l[j-1][j] * u[j][j-1]);

        // u_i,i+1 = a_i,i+1 / l_i,i
        u[j+1][j] = a[j+1][j] / l[j][j];

    } // for loop

    // Step 3: set l_n,n-1 = a_n,n-1
    l[nNode-2][nNode-1] = a[nNode-2][nNode-1];

    // set l_n,n = a_n,n - l_n,n-1 * u_n-1,n
    l[nNode-1][nNode-1] = a[nNode-1][nNode-1] - l[nNode-2][nNode-1]*u[nNode-1][nNode-2];

    // Step 4 and 5 solves for lz=b
    // Step 4: z_1 = a_1,n+1 / l_1,1
    z[0] = a[nNode][0] / l[0][0];

    // Step 5
    // i=2,...n
    for (int j = 1; j < nNode; j++)
    {
        // z_i = (1/l_i,i)*[a_i,n+1 - l_i,i-1 * z_i-1]
        z[j] = (1/l[j][j])*( a[nNode][j] - l[j-1][j]*z[j-1]);

    } // for loop

    // Step 6 and 7 solve ux=z
    // Step 6: x_n = z_n
    x[nNode-1] = z[nNode-1];

    // Step 7: i=n-1,...1
    for (int j = nNode-2; j > -1; j--)
    {
        // x_i = z_i - u_i,i+1 * x_i+1
        x[j] = z[j] - u[j+1][j] * x[j+1];

    } // for loop

} // croutSolver
```

Figure 25 Java implementation of a tridiagonal matrix solver

The Java implementation for calculating the coefficients for the interior nodes for matrix a[][] is shown in Figure 26.

```

public void calcInteriorNodes(double a[], double b[], int nNode, double ambientTemp,
                           double nSqr, double deltaX)
{
    // calculates the coefficients for the a[] matrix - interior nodes of the strip.
    // Initialise variables

    double aWest = 0.0;
    double aEast = 0.0;
    double sPoint = 0.0;
    double aPoint = 0.0;
    double sU = 0.0;

    // Calculate coefficients
    aWest = 1/deltaX;
    aEast = 1/deltaX;
    sPoint = -nSqr*deltaX;
    aPoint = aWest + aEast - sPoint;
    sU = nSqr*deltaX*ambientTemp;

    // we now initialise the a[] matrix with zeros across the board
    for (int j = 1; j < nNode; j++)
    {
        for (int i = 0; i < nNode; i++)
        {
            a[i][j] = 0.0;
        } // nested for loop
    } // outer for loop
    // Now we place the coefficient values in a[]
    for (int j = 1; j < nNode - 1; j++)
    {
        // we make aWest and aEast negative bcos we bring them over to the
        // LHS of the equation aPoint = aWest + aEast + sU
        a[j-1][j] = -aWest;
        a[j][j] = aPoint;
        a[j+1][j] = -aEast;
        b[j] = sU;
    } // outer for loop
} // calcInteriorNodes

```

Figure 26 Java routine to calculate coefficients for interior nodes

The java code for the two end point boundaries are similar to the one in Figure 26, except they have different coefficient values to the east and west nodes as shown in Figures 27 and 28 respectively.

```

public void calcWestBoundary(double a[], double b[], int nNode, double ambientTemp,
                           double nSqr, double deltaX, double bTemperature)
{
    // Initialise variables
    // double aWest = 0.0;
    double aEast = 0.0;
    double sPoint = 0.0;
    double aPoint = 0.0;
    double sU = 0.0;

    // Calculate coefficients
    // aWest = 0.0;
    aEast = 1/deltaX;
    sPoint = -nSqr*deltaX - ((2.0)/deltaX);

```

// cont.

```

aPoint = aEast - sPoint;
sU = nSqr*deltaX*ambientTemp + ((2.0)/(deltaX))*bTemperature;

// Now we place the coefficient values in a[][]
a[0][0] = aPoint;
a[1][0] = - aEast;

for (int i = 2; i < nNode; i++)
{
    a[i][0] = 0.0; // zero out the other elements on the same row
}

// enter the sU value in the corresponding b[] matrix
b[0] = sU;

} // calcWestBoundary

```

Figure 27 Java routine to calculate coefficients for the west node

```

public void calcEastBoundary(double a[], double b[], int nNode, double ambientTemp,
                           double nSqr, double deltaX)
{
    // Java routine to calculate coefficients for the east node
    // Initialise variables
    double aWest = 0.0;
    //double aEast = 0.0;
    double sPoint = 0.0;
    double aPoint = 0.0;
    double sU = 0.0;

    // Calculate coefficients
    aWest = 1/deltaX;
    // aEast = 0.0;
    sPoint = -nSqr*deltaX;
    aPoint = aWest - sPoint;
    sU = nSqr*deltaX*ambientTemp;

    // Now we place the coefficient values in a[]
    a[nNode-1][nNode-1] = aPoint;
    a[nNode-2][nNode-1] = - aWest;

    for (int i = nNode - 3; i > -1; i--)
    {
        a[i][nNode-1] = 0.0; // zero out the other elements on the same row
    }

    // enter the sU value in the corresponding b[] matrix
    b[nNode-1] = sU;

} // calcEastBoundary

```

Figure 28 Java routine to calculate coefficients for the east node

2.3.4 Algorithm for diffusive heat transfer in 1d

In order to calculate the internal temperature of the heat shield, we need to know the external temperature. The external temperature depends on the distance away from the blast of the heat shield. The heat shield is resident in the first horizontal strip; all that needs to be done is to find out which control volume the heat shielding lies within. To do this we use a search algorithm to identify the correct control volume and corresponding temperature of that control volume. Figure 29 illustrates the control volume boundaries and temperatures for a four-node system.

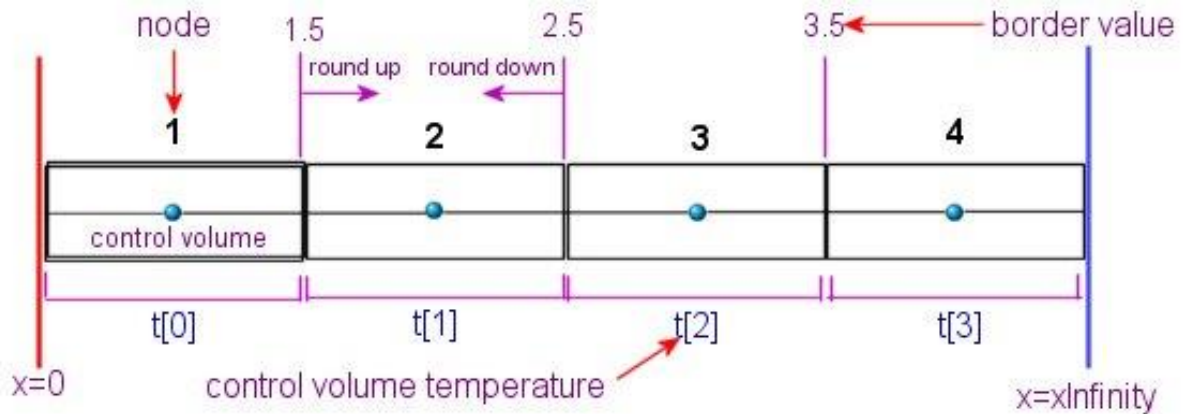


Figure 29 Temperature range for control volume cells

Figure 30 represents the java implementation to determine the external temperature of the heat shielding.

```
public double externalVesselTemp(double t[], double xVessel, double xInfinity, int nNode)
{
    /* We need to find out what control volume the vessel lies in in order that the
    node may be identified and thus the corresponding temperature attained */

    // initialise variables
    double externalTemp = 0.0;
    double node = 0.0;
    double remainder = 0.0;

    // Calculate position of vessel in terms of the nodal points
    node = (xVessel/xInfinity) * nNode;

    // Identify the decimal part of the number "node" to see which boundary it is nearer to.
    remainder = node - Math.floor(node);

    if (remainder >= 0.5)
        node = Math.ceil(node); // round up to highest node point
    else
        node = Math.floor(node); // round down

    /* Next we need to identify whether the node lies on the end points of the strip, if so we have to round up or down
    depending on which end of the strip it is on. */

    if (node == 0.0) node = 1.0; // West face correction

    if (node == (nNode + 1)) node = nNode; // East face correction

    // We now identify the temperature associated with that control volume at "node"
    return stripTemp[(int)(node-1)];
} // externalVesselTemp
```

Figure 30 Java algorithm used to determine the external temperature of the heat shielding material

Once the external temperature is obtained, the heat flux passing through the heat shielding needs to be known. This is calculated using Equation 16 below.

$$q_{conv} = h_c A (T_B - T_V)$$

Equation 16 Heat flux passing through the heat shield

Where:

q_{conv} - heat flux passing through a surface area A of the air strip / J s^{-1} or W

A - cross-sectional area of air strip / m^2

T_B - boundary temperature of horizontal strip / K

T_V - external vessel temperature (previously determined) / K

h_c - convective heat transfer coefficient of air, which can assume to be $30 \text{ W K}^{-1} \text{ m}^{-2}$ (at 300K)

The Java implementation of the code is given in Figure 31.

```
public double heatFlux(double crossSecA, double boundaryTemp, double extVesselTemp,
    double hConv)
{
    double qConv = 0.0;
    qConv = hConv * crossSecA * (boundaryTemp - extVesselTemp);
    return qConv;
} // heatFlux
```

Figure 31 Heat flux algorithm across heat shielding

Now the final calculation: the interior temperature of the heat shielding. Fourier heat diffusion equation is used to determine the interior temperature, shown in Equation 17 and Figure 34.

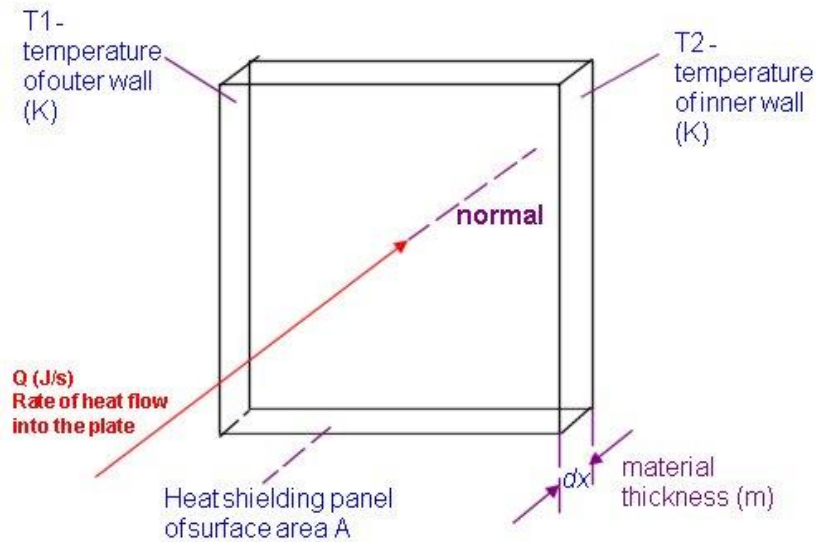


Figure 34 Diffusive heat transfer on a heat shield plate

$$t_{int} = t_{external} - \frac{q_{conv} x_{thickness}}{kA}$$

Equation 17 Fourier heat diffusion equation

Where:

t_{int} - interior temperature of heat shielding / K

$t_{external}$ - external vessel temperature / K

q_{conv} - heat flux across surface A (Q in Figure 34) / J s^{-1}

$x_{thickness}$ - thickness of heat shielding material / m

k - thermal conductivity of heat shielding material / $\text{W m}^{-1} \text{ K}^{-1}$

A - cross-sectional area of heat shielding / m^2

Figure 32 represents the Java implementation of the Fourier diffusive heat transfer equation.

```
public double interiorTemperature(double extTemperature, double qConv,
                                double xThickness, double kValue, double materialArea)
{
    double intTemperature = 0.0;

    intTemperature = extTemperature - ((qConv)*(xThickness))/((kValue)*(materialArea));
    return intTemperature;

} // interiorTemperature
```

Figure 32 Algorithm for Fourier heat diffusion across a plate

Having obtained the internal temperature of the heat shielding plate, all that is left to do is to compare this result with the threshold temperature of the vessel: $393\text{K} \pm 10\text{K}$ ($120^\circ\text{C} \pm 10^\circ\text{C}$). If the design succeeds, the final result will be displayed, but if the design fails, the computer program will recommend an appropriate thickness that is under the 10 inches limit. If no thickness can be found for the particular experiment the computer program will recurse through the material database to find an appropriate replacement for the failed material. The algorithm in Figure 32 finds the optimum thickness for a particular material.

```
public double optimumThickness(double width[], double temperature[], double extTemp,
                               double qConv, double kVal, double crossSecA,
                               double thresholdTemp)
{
    // Initialise variables
    double optThickness = 0.0;
    double xThickMeters = 0.0;
    double intTemp = 0.0;

    /* increment the thickness of the heat shielding plate 0.5 inches each time
       and calculate the corresponding internal temperature. If the internal temperature
       is less or equal to the threshold temperature exit loop */

    for (int i = 0; i < 20; i++)
    {
        xThickMeters = xThickMeters + 0.0127; // 0.0127m = 0.5 inches

        // calculate interior temperature
        intTemp = interiorTemp(extTemp, qConv, xThickMeters, kVal, crossSecA);

        // record results
        width[i] = xThickMeters;
        temperature[i] = intTemp;

        if (intTemp <= thresholdTemp)
            return xThickMeters;
        else
            return -1.0; // i.e. no thickness qualifies

    } // for loop

} // optimumThickness
```

Figure 33 Java routine to find the optimum heat shielding thickness

2.3.5 Computational limitations

Because of the large number of arrays used in the calculations, the array size is very important. We could just use one large array of maximum size and utilise what fraction of it's space that we need, but due number of arrays being used, that would not be feasible due to computer memory size restrictions. Therefore, to overcome this problem, an **if** statement is introduced at the time of post-processing, whereby the user enters the number of nodal points and the **if** statement would initialise an appropriate array for that particular size. For example, if the user chooses to create a mesh with only 5 nodes, there would be no point using an array the size of a[50000][50000] ! Thanks to the object-orientated nature of Java, all arrays are objects that can be initialised at any point in the program. Example the array to hold the fallout region temperatures, falloutMatrix, would be declared as follows:

```
private int yres = 5;
private int nNode = 5;
private double[][] falloutMatrix; //fallout region matrix
```

Note: the dimension of the array have a set minimum size, this can be changed to the desired value anywhere after declaration. The matrix is then initialised thus:

```
falloutMatrix = new double[nNode][yres];
```

The values yres and nNode can be changed allowing for a matrix of that specific size to be initialised. This saves enormous amounts of time and memory when considering very small arrays and extremely larger arrays.

3 Software Design: Optimus Prime

3.1 Requirements Analysis

The software to be designed is to meet a set criteria laid down by the physicists at research facility. A bespoke application is needed to perform a series of calculations to test if a material used for heat shielding is suitable for use on a unmanned military vehicle exposed to high temperatures generated by a nuclear blast. The operations and features of the software are as follows:

- **Add a new warhead / heat shielding material to database**
- **Remove existing warhead / heat shielding material from database**
- **Modify existing warhead / heat shielding material from database**
- **User-friendly pre-processing wizard (The user should be able to select the parameters needed to perform the experiment from data held on a remote database: The user should be able to modify the parameters from the database should he need to tweak the results, i.e. the input parameters should be kept flexible, not confined strictly to the database values.)**
- **Comprehensive graphic output of the results and fireball + fallout regions**
- **Two modes of heat transfer: Diffusion and Convection-Diffusion (Both steady state 2-Dimensional flow)**
- **JPEG format output of the results - This should hold all the results as well as the relevant information representing the parameters used.**
- **Fully Object Oriented and Platform Independent**
- **The system should be easy to setup and maintain**
- **Overall user-friendly interface, which should be kept simple and intuitive.**
- **Seamless database integration: Databases can be changed without affecting the code**
- **Graphical output of the solution trend for one strip to see if convergence is satisfied (Convection-Diffusion mode only)**
- **The program should have the option to increment the thickness of the heat shielding if it fails the initial test**
- **The program should have the option to seek alternative materials from the database if the primary material fails**
- **The user should have the option to amend the experimental meshing parameters at the processing stage, if he/she is not satisfied with the results: the meshing resolution should be**

Two additional features of the system, Optimus Prime, are Thermal Image Amplification and Diffuse Fallout Region. The former is that if the fallout region has temperatures too small compared to the fireball region it doesn't show up on the thermal graph, thus this function recalibrates the thermal image key only using the temperatures in the fallout region, this allows all the temperatures in the fallout region to be displayed. The latter function simply averages out the

exiting temperatures using a Jacobi averaging algorithm, similar to the one used for the fireball region. Averaging simply yields a more realistic thermal model of the fallout region.

The chosen implementation language for the system was Java (Sun JDK 1.3.1_03) and was developed using Borland JBuilder 6 Enterprise and Designed in Rational Rose Enterprise Edition. The database used was MS Access 2000.

3.2 DATABASE DICTIONARY

The database holds five tables: WARHEAD, ATM_PROPERTIES, CONSTANTS, SAFETY_RANGE, and SHIELDING. Tables WARHEAD and SHIELDING can be left empty as they represent the nuclear warhead and heat shielding material respectively. The design for SHIELDING is given below in table 10.

Field	Type	Description
materialID	int	Unique key to identify the material in the database
materialName	String	The name of the heat shielding material
kValue	double	The thermal conductivity of heat shielding
materialThickness	double	Default value = 0.0125m (= 0.5 inch)
materialDensity	double	The density of the heat shielding material
crossSectionA	double	The cross sectional area of the material / m ²
cValue	double	Specific heat capacity of heat shielding

Table 10 Fields in SHIELDING table

The other three tables are constant, as they are used only for reference purposes by the program: ATM_PROPERTIES holds two fields: **altitude** (Distance from the surface of the earth to the point of blast / m) and **airDensity** (The density of air at the corresponding altitude / kg m⁻³) this table is used to determine the air density at the HOB, this allows the program to calculate the temperature of the fireball for a given internal energy, which is dependent on the air density. The CONSTANTS table contains the physical properties of the air; Table 11 shows the values of the constants.

Field	Type	Description
SHC_AIR	double	Specific heat capacity of air (default value = 1.00e3 J kg ⁻¹ K ⁻¹)
ambientTemperature	double	Ambient temperature of air before detonation (default value = 293K)
CHT_AIR	double	Convective heat transfer coefficient of air (default value = 30 W m ⁻² K ⁻¹ @ 300K)
k_AIR	double	Thermal conductivity of air (default value = 0.025 W m ⁻¹ K ⁻¹ @ 293K)

Table 11 Fields in CONSTANTS table

The final table, SAFETY_RANGE, contains the reference to the nuclear warhead yield (kT) and the corresponding radius of minimum thermal effect (m). This is the distance where the thermal damage from the bomb is negligible. The program uses this value for x_infinity. These values extracted from data in table 7 above.

The database is put up as an ODBC resource, the client program, Optimus Prime, accesses the database using a JDBC-ODBC java driver along with SQL to query the database. The database is held on a remote server and can be accessed by clients on the LAN network. Centralising the database like this ensures that any change in the WARHEAD or SHIELDING tables are reflected in the whole system with minimum administration across all the clients.

3.3 Use Case Modelling

The use cases are derived from the user requirements; they capture and communicate the user requirements in graphical format. This is the external model of the system. From the use case diagram, we can identify the boundary of the system, the users that interact with it, and the main modules of functionality of the software. The use case diagram for derived for Optimus Prime is shown in Figure 34, it is clear from the diagram that the system is divided into two parts: one is the experimental part of the program, the other is used purely for the administration of the system. (database management). Each use case represents one complete process that the system must perform in order to benefit the user. The use cases procured from the system requirements are:

- **Pre-processing** – Collects all the necessary information from the user to run the experiment
- **Run Experiment** – Runs the experiment and displays the results
- **Load Results** – Loads results and displays them in graphical format
- **Save Results** – Saves the results to JPEG format as well as a meta file

- **System Setup** – Sets up the database DNS, username and password. A Setup.ini file is created
- **Add Warhead** - Adds new warhead record to WARHEAD table on the remote database
- **Modify or Remove Warhead** – Maintenance of records in WARHEAD table held on remote database
- **Add Material** – Adds new heat shielding material record to SHIELDING table on the remote database
- **Modify or Remove Material** - Maintenance of records in SHIELDING table held on remote database

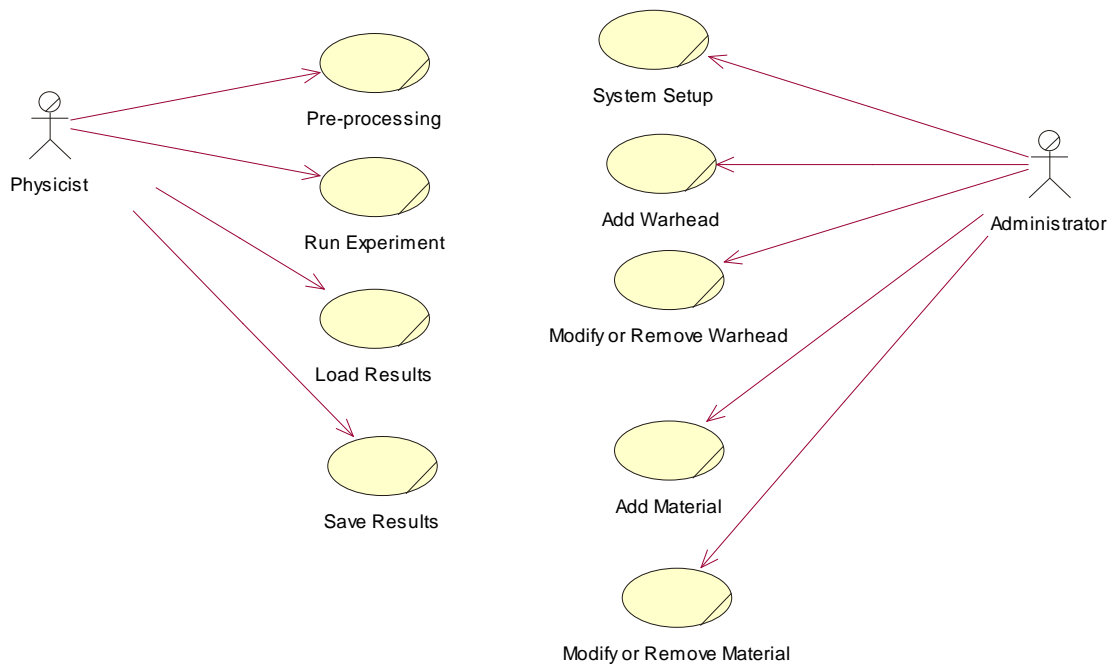


Figure 34 USE CASE diagram for Optimus Prime version 1.0

3.4 Domain Analysis: Conceptual Class Diagrams

The domain classes (concept classes) in Figure 37 represents the key business classes that make up the core system design. Business classes simply mean that they handle the main processes that are involved in carrying out the primary function of the program. These domain classes are drawn from the user specifications, along with the methods and attributes of the classes. The domain class diagram gives a rough initial outline of the system, which is modified and enhanced in the final design. The domain classes in Optimus Prime are as follows: **warhead, fireball, diffusion, cell, solver, strip, fallout, vessel, shielding and material**. The association between the classes is outlined with the connected lines. The upper segment of the class box signifies the attributes of the class and the bottom half the methods that the class is capable of performing. For the warhead class, we can see from the diagram that it holds information about the yield of the bomb, the name, type, thermal output and efficiency of the weapon as well as a unique id to identify the weapon on file. The methods of warhead simply outline the functions of the class, e.g. setName() sets the name of the name attribute, calcThermalEnergy() returns a double value signifying the thermal output in joules of the weapon, derived from the yield and efficiency.

3.5 User interfaces and sequence diagrams for the Pre Processing and Run Experiment

A scenario is a particular path through a use case. An actor always instigates the scenario; in this case it is the physicist, who performs an action on the system that runs the experiment. A Sequence diagram displays the path through one scenario of a use case. In this chapter the two main sequence diagrams that will be built are: Pre-processing and Run experiment use cases. The scenario for the pre-processing use case is as follows:

3.5.1 USE CASE: Pre-Processing

Using the sequence diagrams a pre-processing wizard was constructed to collect the data from the user in a GUI interface that checks that all that all the parameters entered are within acceptable bounds. These boundaries are outlined

below along with the associated user interface. The sequence diagram for this Use Case is given in Appendix A due to the large size it could not be displayed on these pages.

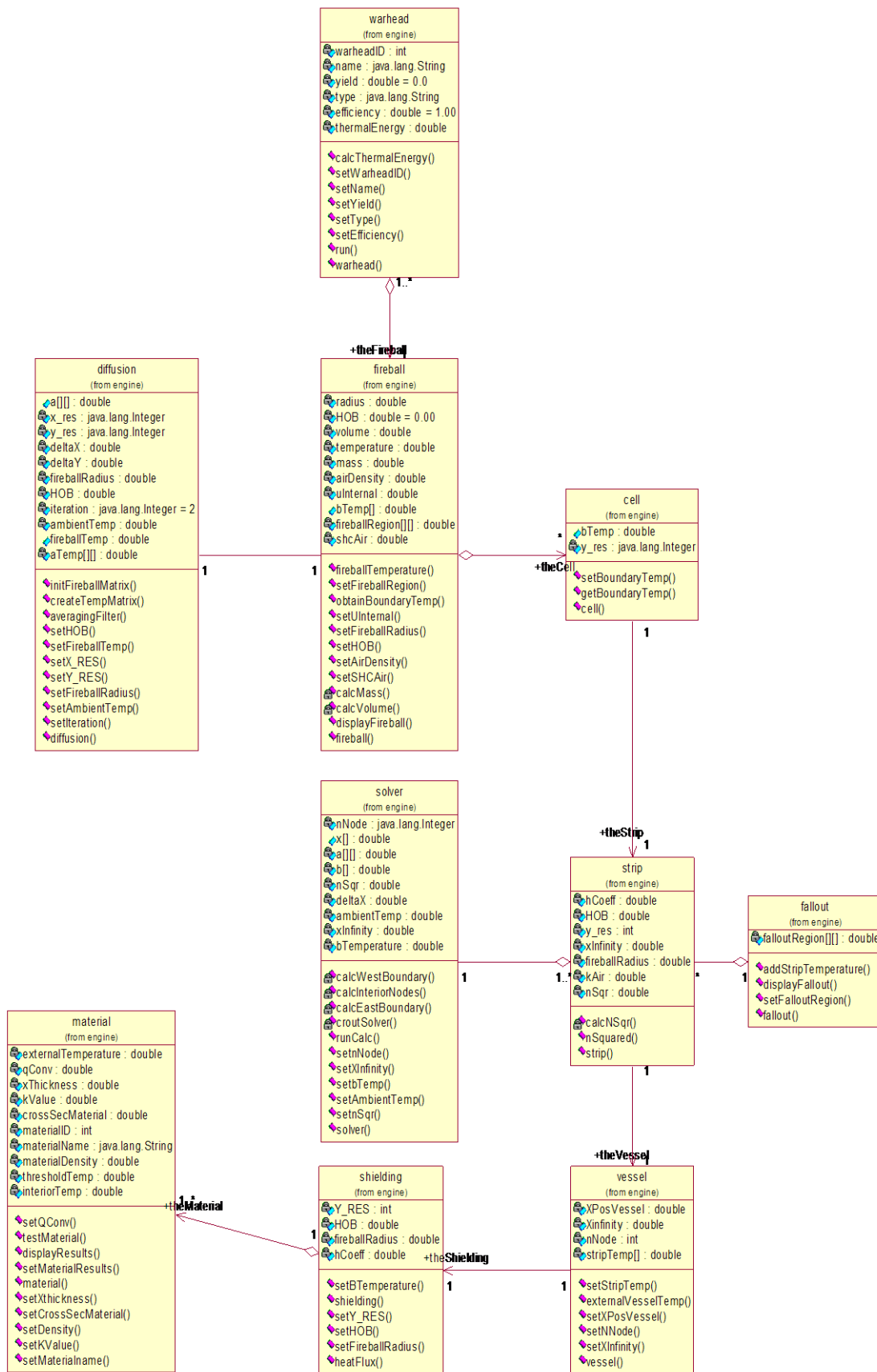


Figure 37: Domain class structure for Optimus Prime

a) Select missile from database

Physicist selects a WARHEAD from a list, where the following information of the selected warhead is displayed in a wizard window for the user to modify and commit the changes as shown in Table 12 and Figure 35.

Warhead Properties	Type	Range
Name	String	N/A
Type	String	Implosion assembly / gun assembly / layered sphere
Efficiency	double	0.00 – 1.00
Yield	double	1 – 60,000 kT
HOB	double	0 – 30,000 m
Fireball Radius	double	100 – 10,000 m

Table 12 Variables for selecting missile from the database

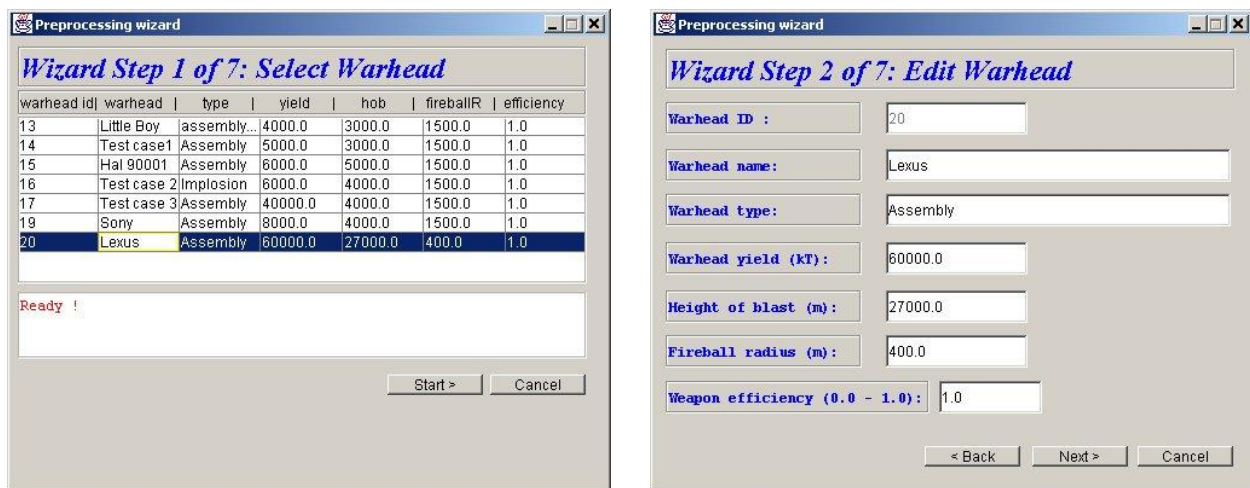


Figure 35 GUI interface for selecting and editing a missile from database

b) Set environmental parameters

Constants representing the environmental properties are displayed in textboxes for the user to modify and commit. The constants are read in from the remote database into the interface. The parameter range brackets and interface are shown in Table 13 and Figure 36 respectively. When the user clicks “Next >” the system validates the form for typing errors and checks to see if the values are within the acceptable ranges.

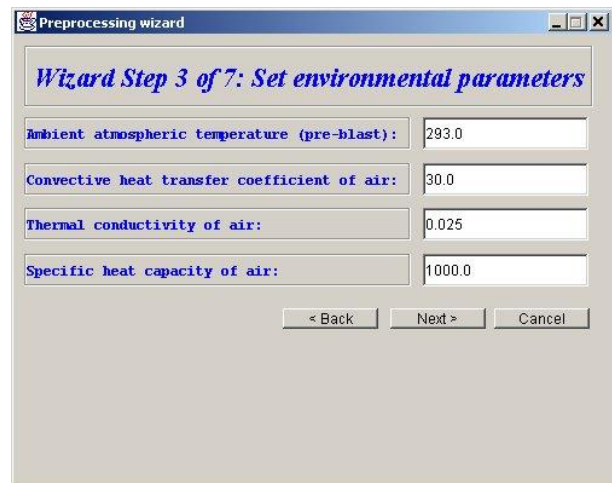


Figure 36 User interface for setting environmental

Environmental parameters	Type	Range
Specific heat capacity of air	double	$0.5 - 1.5 \times 10^3 \text{ J kg}^{-1} \text{ K}^{-1}$
Ambient atmospheric temperature	double	223 - 333 K
Convective heat transfer coefficient of air	double	$10 - 50 \text{ W m}^{-2} \text{ K}^{-1}$
Thermal conductivity of air	double	$0.005 - 0.09 \text{ W m}^{-1} \text{ K}^{-1}$

Table 13 Variables and their respective ranges for setting environmental parameters

c) Set meshing parameters

The user sets the following parameters representing the resolution and refinement of the mesh over the two regions: Fireball and Fallout. The variable range and user interface is shown in Table 14 and Figure 37 respectively.

Meshing parameters	Type	Range
Fireball region: x - resolution	int	Must be greater than 4
Fireball region: y - resolution	int	Must be greater than 4
Fallout region: number of nodes	int	Must be greater than 4
Number of iterations	int	Minimum number of iterations = 2

Table 14 Meshing variables and their respective ranges

Figure 37 Wizard user interface for setting the meshing parameters

d) Set vessel parameters

The user is prompted to set the two main properties of the target vessel: it's horizontal distance from the blast, meaning the distance from the fireball boundary to the vessel in meters, and the second parameter is the maximum threshold temperature; this is the maximum operating temperature for the computer equipment held within the target vessel. The ranges are given below in Table 15 along with the user interface in Figure 38.

Figure 38 Wizard interface for setting the vessel parameters

Vessel parameters	Type	Range
Horizontal vessel distance from blast	double	Greater than zero meters
Threshold temperature	double	Operating temperature should be between 223 - 393 K

Table 15 Vessel parameters and their respective range

e) Select heat-shielding material from database

The physicist selects a material from the database to be tested on the vessel. The material properties of the heat shielding material are read in from the database and modified via the user interface as shown in Table 16 and Figure 39 respectively. After the user commits the changes, all the data is written to file names “preprocess.dat” ready to be utilised by the run experiment function. A summary is displayed after the file is written (Appendix A.2).

Heat shielding material properties	Type	Range
Name	String	N/A
Thermal conductivity	double	0.05 - 1.0 J s ⁻¹ m ⁻¹ K ⁻¹
Thickness	double	0.0125 - 0.254 meters
Density	double	1000 - 3000 kg ⁻¹ m ³
Specific heat capacity	double	1500 - 3000 J kg ⁻¹ K ⁻¹
Cross sectional area	double	0.01 - 0.05 m ²

Table 16 Heat shielding parameters and their respective ranges

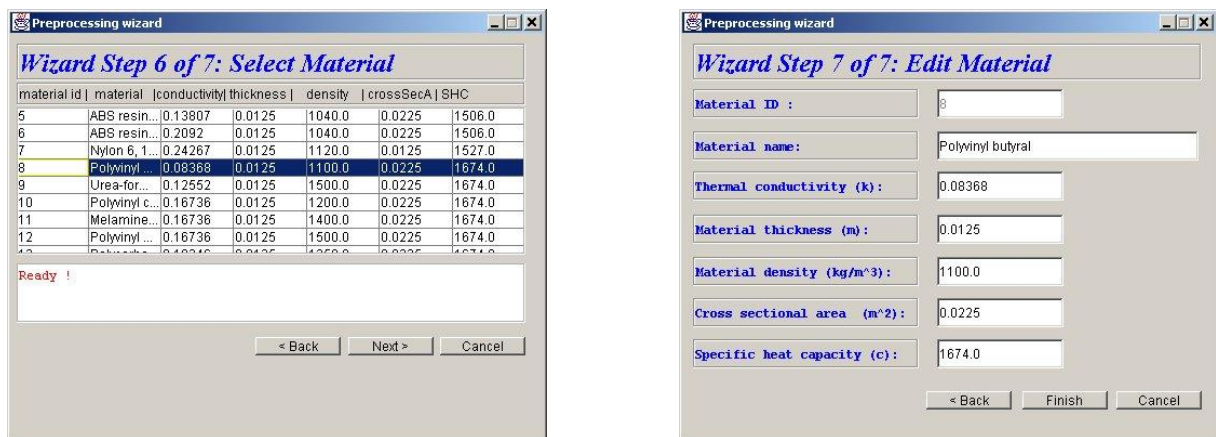


Figure 39 User interface for selecting and editing the heat shielding material

3.5.2 Use Case: Run Experiment

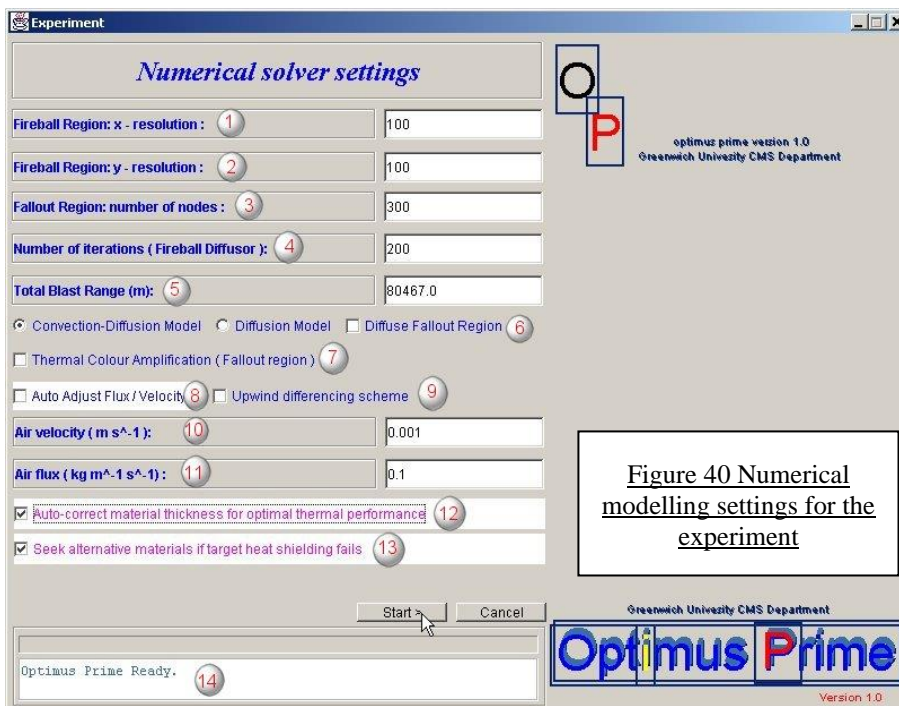


Figure 40 Numerical modelling settings for the experiment

After the pre-processing file has been written, the user can then be able to run the experiment based on the parameters set in the pre-processing wizard. The Run Experiment function allows the user to “tweak” the settings found in the pre-processing file. Referring to Figure 40 the options available to the user are examined.

1. Fireball Region: x – resolution

This value is read in directly from file, the user can change the mesh resolution across this area to make the solution across this region coarser to yield better results.

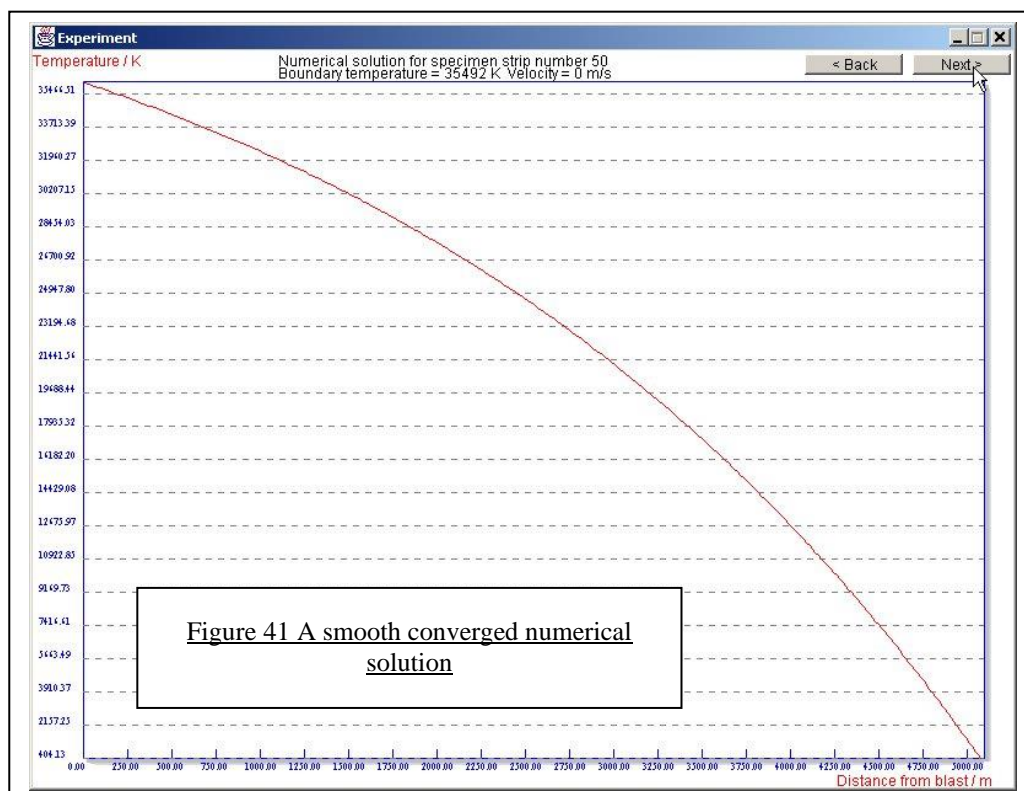
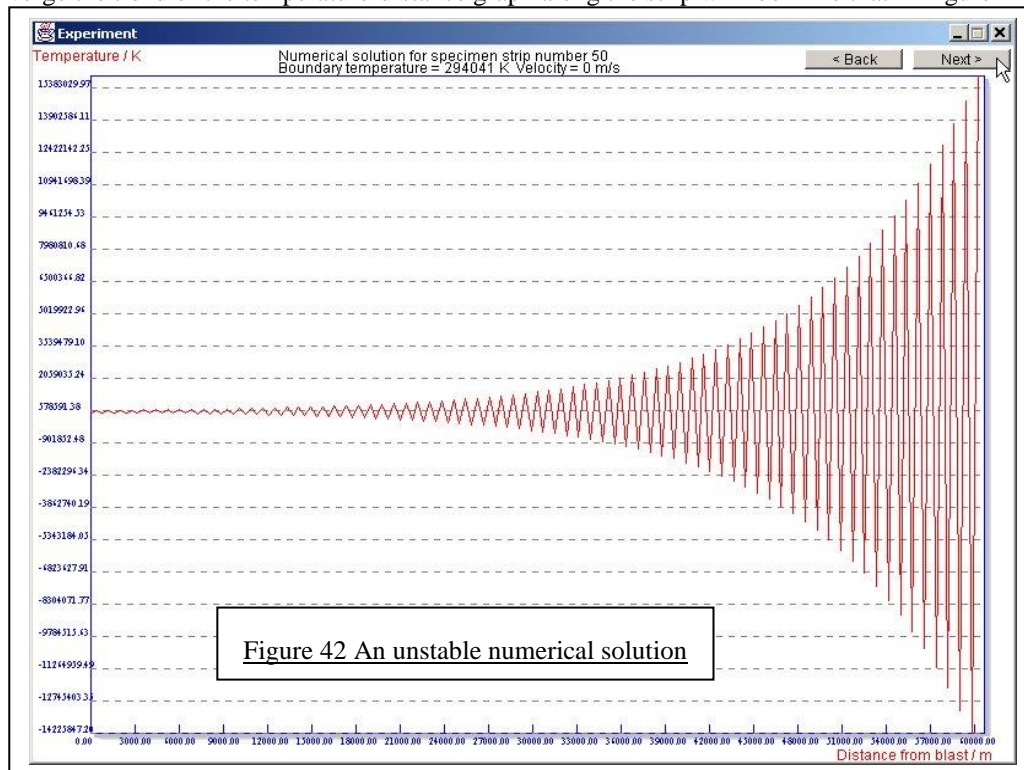
2. Fireball Region: y – resolution

This value is also read directly

from file, this represents the mesh resolution in the vertical plane, this also is the resolution of the fallout region as the two are connected. Increasing this value, increases the number of strips in the fallout region as well the number of divisions in the fireball region. The user can tweak this value to yield better results at the expense of computer memory, as it creates a larger $n \times m$ matrix.

3. Fallout Region: number of nodes

As shown in Figure 21, this value represents the number of control volumes across each strip. If the solution does not converge the trend of the temperature-distance graph along the strip will look like that in Figure 42



The diagram in Figure 41 shows a convergent solution, which is a smooth function. Successful solutions depend on the number of nodes and the resolution of the mesh.

4. Number of iterations: Fireball Diffuser

This value signifies the number of times the averaging filter should be run over the fireball region to “diffuse” the fireball temperatures over the ambient temperature. The higher this number, the more distributed the fireball temperature would be, and hence a better spread across the fireball region of the fireball temperature. If this value is made too small (less than 2 say) the fireball temperatures would be concentrated in the fireball sphere, thus not allowing an even distribution to be established. Making this value too great for large matrices will take enormous amounts of computing time. The user can tweak this value to obtain a balance of speed and accuracy.

5. Total Blast Range (m)

This value represents x_{∞} in the calculations, and is obtained from a table on database using the weapon’s yield as reference. This value is the radial distance of negligible thermal effects due to the explosion. The user can change this value to observe the thermal effects over a shorter range.

6. Convection-Diffusion Model / Diffusion Model & Diffuse Fallout Region

Diffusion Model: Here we apply the solver outlined in the previous chapter to solve the steady state diffusion in 1D (i.e. for one strip) and amalgamate all the strips that make up the fallout region. Air velocity from the blast point outwards does not apply to this model, so the thermal energy will only be transported by diffusion no other influences, thus the thermal effects will be of much smaller range than that of the convection-diffusion model. The user may want to decrease the Total blast range settings to magnify the thermal image in the post-processing phase of the experiment.

Convection-Diffusion Model: Here we have a dual mode energy transportation: the diffusion model mentioned above as well the convection model which has two extra elements thrown into the boundary coefficient calculations: the air flux ($\text{kg m}^{-1} \text{s}^{-1}$) (that is the amount of air passing through a plane which is perpendicular to the velocity direction) and the air velocity (m s^{-1}). Enabling this engine solver will require the user to input the values for the air flux and air velocity signified by numbers 10 and 11 in Figure 40. The code for this algorithm is given in Appendix B.1. A successful solution is determined by dividing the air velocity by the flux, this yields a ratio known as the Pe ratio where the condition for convergence is: $Pe < 2$. If Pe is greater than 2 then we get an unstable result for temperature across the strip as shown in Figure 42. There are three other options available to the user if the solution is not found they are;

- 1). Use the Upwind Differencing Scheme: this is an algorithm specifically used for high Pe values. (shown as number 9 in Figure 40) - The code for this algorithm is given in appendix B.2.
- 2). Enable the Auto-Adjust toggle (shown as number 8 in Figure 40). This option increases the Air Flux value and simultaneously decreases the Air Velocity value until $Pe < 2$. The code snippet for this function is shown in Figure 43.

```
// increase the flux value until Pe < 2
if (autoAdjustFlux == true) {
    while (Pe > 2) {
        flux = flux + 1e-15;//0.0000000001; // increase flux slightly
        airvelocity = airvelocity - 1e-3;//0.001; // decrease air velocity slightly
        convDiff.setFlux( flux );
        convDiff.setAirVelocity( airvelocity );
        convDiff.initConstants();
        Pe = convDiff.check4Convergence(); // check Pe for convergence
    } // while
}
```

Figure 43 Code to auto - adjust flux/velocity to achieve convergence

- 3). The last option would be to adjust the velocity or flux value manually as well the Total Blast Range, as it affects the value of deltaX in the main numerical solver.

Diffuse Fallout Region: This function simply passes the fallout region matrix to the averaging filter. The purpose for this would be to smooth out the thermal effects around that region to give more realistic results. The number of

iterations is taken from the same value for the Fireball Region Diffuser (shown as number 4 in Figure 40). This can add a lot to the computational load, leaving the system locked in calculation for longer than 20 seconds on a normal Pentium II 450 Mhz processor with a 300 by 200 Fireball Matrix and 200 by 400 Fallout matrix (200 iterations).

7. Thermal Colour Amplification (Fallout Region)

This switch enables comparatively small temperatures to be amplified. Normally, the temperature key for the fallout and fireball region is the same. The key is generated by a piece of code that simply calibrates itself using the maximum and minimum temperatures found in the fireball region only. This would mean that the temperature range would be from about the ambient to the fireball core temperature. If the fallout region has temperatures that are small compared to the fireball region, they would not show up on the thermal-graph in post-processing. Thus what the Amplification function would do is to recalibrate the key using the minimum and maximum temperatures only found in the fallout matrix (instead of the fireball matrix). The key would have a label on the bottom specifying that it represent the fallout region if this function is used.

Figure 44 shows the two different keys: Note the temperature scales in both the diagrams. The amplification toggle allows the negligible temperatures to be displayed.

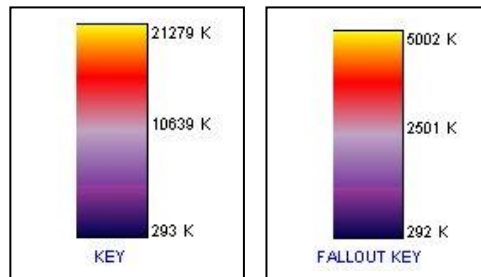


Figure 44 Thermal key for fallout and fireball region

12. Auto-correct material thickness for optimal thermal performance

This function switch adjusts the thickness of the heat shielding material if it fails the experiment. The maximum thickness level is set by 10 inches (0.254 m). The algorithm to test the material for thermal performance is given in Figure 45.

```
public double testMaterial()
{
    double interiorTemp;

    interiorTemp = getInteriorTemp();

    if (autoThickness == false) {
        return interiorTemp;
    } else {

        // if interior temperature is greater than the threshold temp
        // then the test fails, so we must increase the thickness of the
        // material until it is successful ( max. thickness = 10 inch = 0.254 m )

        while ( xThickness <= 0.254 ) {
            interiorTemp = getInteriorTemp();
            if (interiorTemp <= thresholdTemp) break;
            xThickness = xThickness + 0.0127;
        } // while

        return interiorTemp;
    } // if statement
} // testMaterial
```

Figure 45 Java implementation for auto-correct thickness

13. Seek alternative material if target shielding fails

This switch simply looks up the database for alternative materials and runs them all under the same test using the same settings. The results are then displayed in the post-processing phase.

14. Status window

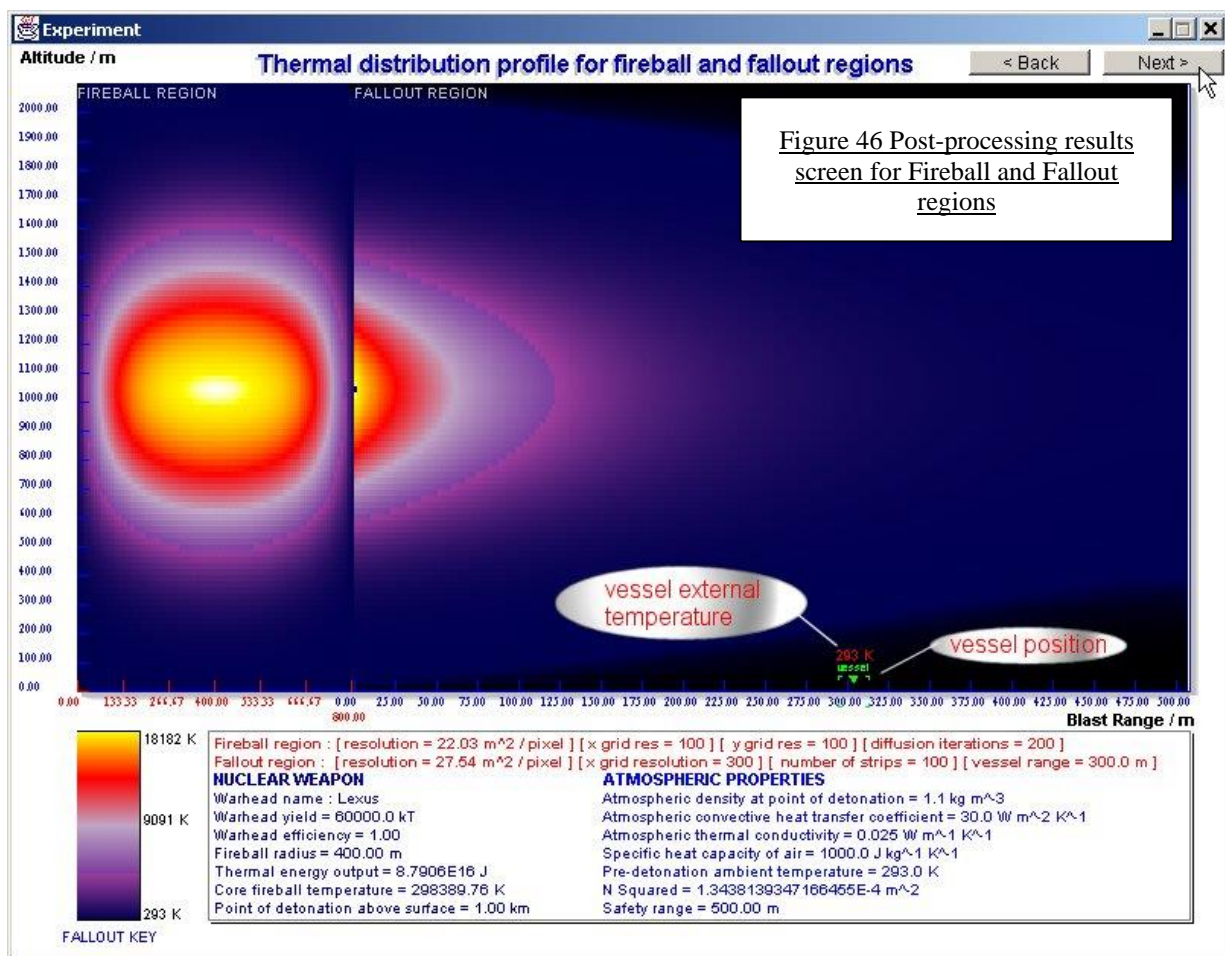
This window displays the current processing status of the experiment along with a progress counter.

Appendix A.3 displays the sequence diagram for the Run Experiment use case.

3.6 Post-processing interface design

In this section we look at the features and construction of the post-processing screens, which displays the results to the user in graphical format. There are three screens, which follow the settings window (Figure 40). The first is the solution trend graph, Figure 41-2, which shows the calculated temperature across one of the strips in the fallout region, namely the strip with the hottest boundary temperature. The graph was designed using the Graphics2D class and all its methods such as drawLine2D and rectangle2D. The graph plots the temperature against the distance; the maximum limit of the x-axis is the distance that is given by the Total Blast Range (xInfinity). The purpose of this solution trend graph is to give the user an idea whether the results make sense or not. The graph is placed on a wizard-type interface, if the user wishes to make any changes to the settings; he/she can simply click “Back” and make the changes, or “Next” to continue to the next window.

After the user clicks “Next” on the interface screen, a number of operations are executed. One of which is the creation of the JPEG image file of the Fireball and Fallout region results screen. This process can take sometime, as the Java Media Framework (JMF), which handles the JPEG compression and creation takes a considerable amount of time to



initialise, and process. The post-processing window of the Fireball and Fallout region is then brought up, which displays the thermal output of the bomb across the two regions. The layout of this screen is given in Figure 46.

All the post processing information and results are sent to the *postProcessor* class to be displayed as a thermal graph. Referring to the graph in Figure 46, the heat from the fireball can clearly be seen on the left and its thermal effects on the right. The key, shown on the bottom left, displays the temperature colour codes for the Fallout region only, because in this case the thermal amplification switch was toggled on, hence the incoherence in the colour between the boundary of the Fireball and Fallout regions. The scale on the y-axis displays twice the altitude of the blast (2*HOB), and the x-axis is divided into two parts: the Fireball scale and the Fallout scale, hence the different colour co-ordination. A small marker on the fallout x-axis represents the vessel position relative to the blast and the temperature above the marker (shown in red) is the external vessel temperature. The information box displays the experimental settings (red), results for the detonation (dark blue), and the atmospheric conditions (light blue).

The thermal image, shown in Figure 46, was constructed using a mesh of rectangles whose size is determined using the x and y resolutions for the Fireball Region and for the Fallout Region: y resolution and n-nodes. Each rectangle, which is scaled to fit the region, is colour coded and displayed using a function called *drawThermalCells*, where each cell is put into a colour band. There are four bands in all, the fireball/fallout matrix is passed as a parameter to this function, where each cell in the matrix will be flagged with an RGB colour code and then assigned a coordinate to be displayed. One of the problems faced using this system, was that the smaller temperature variations were not displayed, for example between the ambient temperature and the lowest fireball temperature and on the other end of the scale the highest temperatures of the core of the fireball where all one colour, and was not distinguishable from the outer core temperatures. To get around this problem one additional colour-bands were introduced to the system, so that more

```
public void drawThermalCells(double x_AxisLength,
    double y_AxisLength,
    int xresolution,
    double[][] fireBM,
    double minTemp, double maxTemp,
    boolean fireballRegion,
    int xBound,
    boolean drawKey,
    Graphics2D gg) {

    double width = (x_AxisLength / xresolution); // sets the width of the cell
    double height = (y_AxisLength / yres); // sets the height of the cell

    int yresolution = yres;
    if (drawKey == true) { yresolution = 1; width = 40.0; }

    int red, blue, green, temp;
    red = 0; blue = 0; green = 0; temp = 0;

    double percentage = 1.0;

    // we employ a four band colour scheme to represent the temperatures

    // First we must divide the Temperature range in four parts, representing the
    // and assign this range to epsilon
    double epsilon = (maxTemp - minTemp) / 4;

    // the epsilon will determine which band the temperature lies within. There
    // are four bands 1, 2, 3, and 4.
    double band1, band2, band3, band4;
    band1 = epsilon + minTemp;
    band2 = band1 + epsilon;
    band3 = band2 + epsilon;
    band4 = band3 + epsilon;

    for ( int j = 0; j < yresolution; j++) {
        for ( int i = 0; i < xresolution; i++) {

            if ( (fireBM[i][j] >= minTemp) && (fireBM[i][j] <= (1.02)*minTemp) ) {

                // coldest region in band1
                percentage = (fireBM[i][j] - minTemp)/((1.02)*minTemp - minTemp);
                red = 0;
                green = 0;
                blue = 0 + (int) ( (80) * (percentage) );
            } else if ( (fireBM[i][j] > (1.02)*minTemp) && (fireBM[i][j] <= band1 - 1) ) {

                // dark blue to purple
                percentage = (fireBM[i][j] - minTemp)/(epsilon);
                blue = 80 + (int) ( (80) * (percentage) );
                green = 0 + (int) ( (63) * (percentage) );
                red = 0 + (int) ( (160) * (percentage) );

            }

        }
    }

    // continued ...
}
```

smaller variations in temperature can also be detected using a very small band values corresponding to the lowest temperatures and highest temperatures on the scale. Figure 47 illustrates the java implementation for the colour band system. The full implementation code for this class can be found in Appendix B.3.

One of the other problems faced during the design of the x-axis scale, was the overlapping of the values which where

```
// continues ...
} else if ( (fireBM[i][j] >= band1) && (fireBM[i][j] <= band2 - 1) ) {

    percentage = (fireBM[i][j] - band1)/(epsilon);
    blue = 160 + (int)( (38) * (percentage) );
    red = 125 + (int) ( (67) * (percentage) );
    green = 63 + (int) ( (102) * (percentage) );

} else if ( (fireBM[i][j] >= band2) && (fireBM[i][j] <= band3 - 1) ){

    percentage = (fireBM[i][j] - band2)/(epsilon);
    red = 192 + (int)( (63) * (percentage) );
    green = 165 - (int) ( (165) * (percentage) );
    blue = 198 - (int) ( (198) * (percentage) );

} else if ( (fireBM[i][j] >= band3) && (fireBM[i][j] <= (0.99)*band4 - 1) ){

    percentage = (fireBM[i][j] - band3)/(0.99*epsilon);
    red = 255;
    green = 0 + (int) ( (255) * (percentage) );
    blue = 0;

} else if ( (fireBM[i][j] >= (0.99)*band4) && (fireBM[i][j] <= band4) ) {
    // hottest region in band4
    percentage = (fireBM[i][j] - (0.99)*band4)/(maxTemp - (0.99)*band4);
    red = 255;
    green = 255;
    blue = 0 + (int) ( (255) * (percentage) );

} // if statement

// print out the thermal scan

// set the colour code...
gg.setColor(new Color(red,green,blue));

if ((fireballRegion == true) && (drawKey == false))
    gg.fill( new Rectangle2D.Double( (i)*width + offset + xOffset,
                                   (j)*height + offset,
                                   width, height ) );

if ((fireballRegion == false) && (drawKey == false))
    gg.fill( new Rectangle2D.Double( (i)*width + offset + xOffset +
                                   xBound,
                                   (j)*height + offset,
                                   width, height ) );

if (drawKey == true)
    gg.drawLine(offset + xOffset,
                (this.getHeight()) - offset - i,
                (int)width + offset + xOffset,
                (this.getHeight()) - offset - i);

} // nested
} // outer for loop
} // drawThermalCells
```

Figure 47 Algorithm for colour coding and displaying the thermal cells

too large to fit between the notches. The way round this was to attenuate the figures to two decimal places and scale the number of notches to a maximum limit of 25 (i.e. there can not be more than 20 notches). A small if statement determines how many digits are in the largest number on the scale and thus reduces the number of notches accordingly so that the numbers can fit without overlapping.

The final post-processing screen that is displayed is the experimental results screen as shown in Figure 48. Here the heat shielding material results are displayed along with the alternative materials (which would be active if the user had toggled the seek alternative materials switch). A small diagram on the top left hand corner shows the heat-shielding panel with the external and internal temperatures. If the heat shielding should fail, the alternative materials found on the

database are run through the same experiment, the ones that fail are displayed in blue and the successful materials are displayed in red. These results are automatically output into JPEG file format under the filename: results.jpg. The previous image files are overwritten. The data from this experiment is saved to disk if the user wishes to load it for later

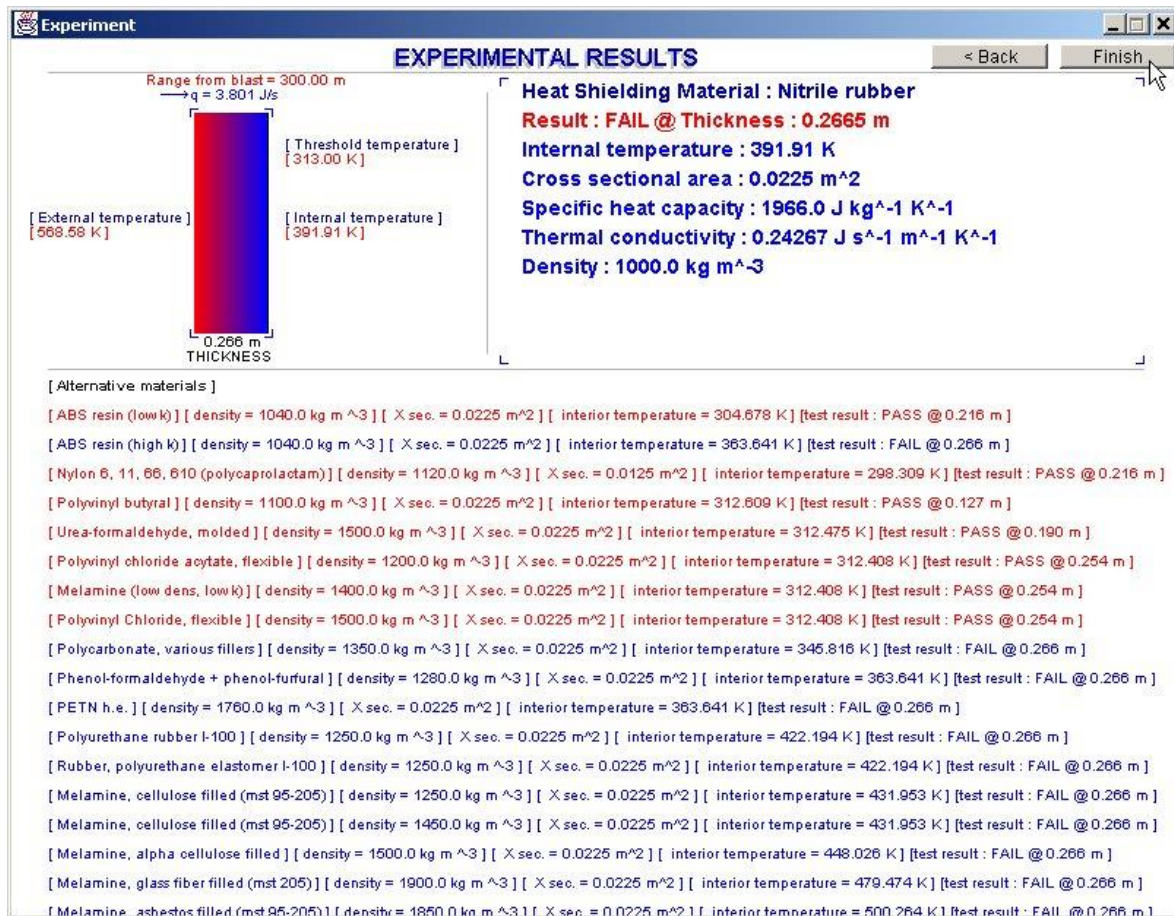


Figure 48 Experiment results screen

use. The user can select his/her own file name, which would be put under the .opt file extension.

3.7 Architecture design

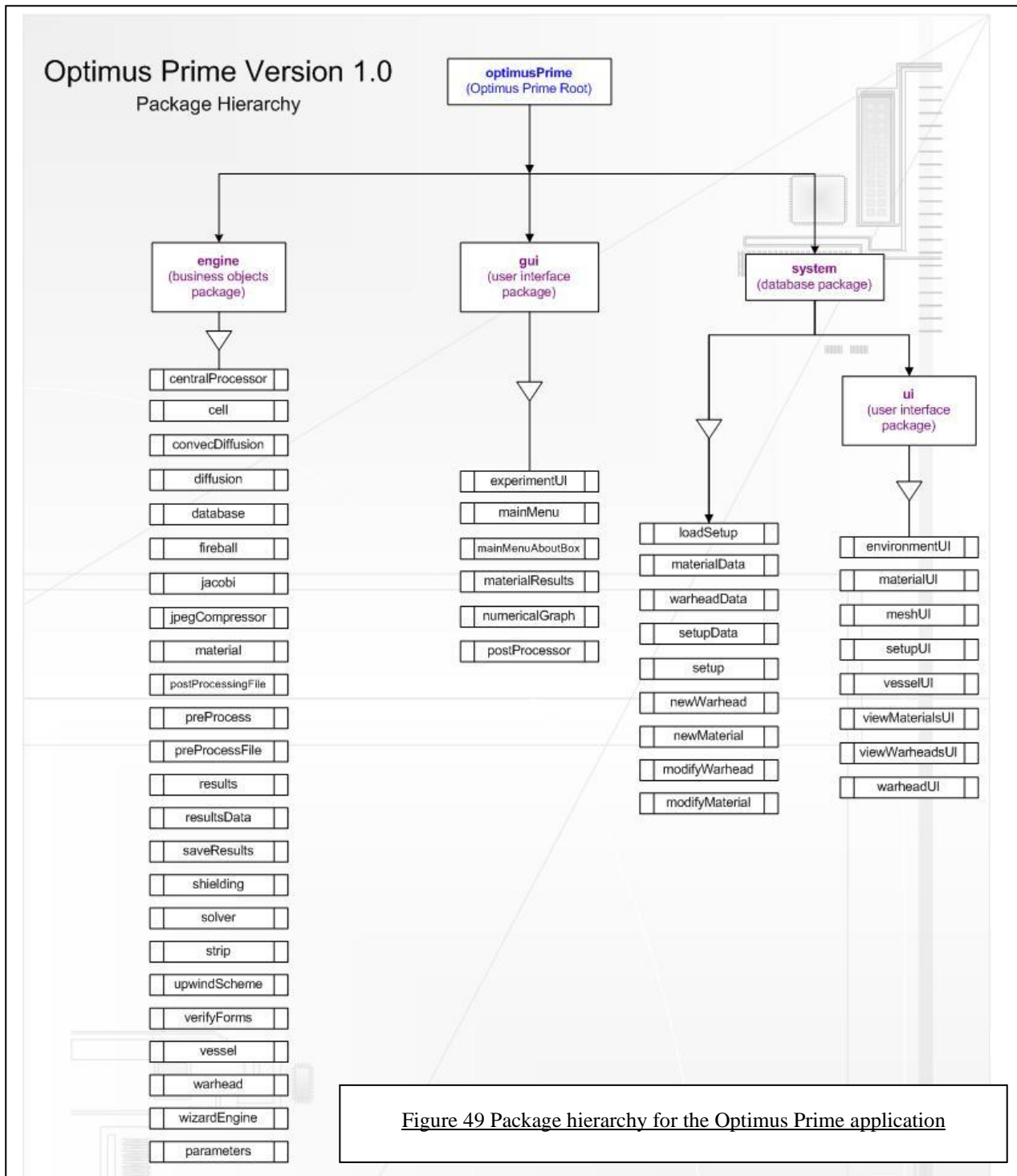
Optimus Prime is structured into directories on disk called packages. The package is separated into different domain classes that make up the application logic, so that it is divided from the technical logic so that changes in either don't impact the other part. A set of rules for dependencies between the packages (e.g., "subsystems") was established so that no bi-directional dependencies are created between packages (in order to avoid packages becoming too tightly integrated with each other). The packages are divided into the following subsystems:

User-Interface Package: ui and gui. These classes are based on the Java AWT package and Java Swing libraries for creating user interfaces. This package cooperates with the Business-Objects package, which contains the classes where the data is actually stored. The UI package calls operations on the business objects to retrieve and insert data into them.

Business-Objects Package: engine. This includes the domain classes from the analysis model such as *warhead*, *material*, *shielding*, *diffusion*, and so on. The design completely defines their operations and adds support for persistence. The business-object package cooperates with the *parameters* object (in the engine class) in that all business-object classes must inherit from the Persistent class in the **engine** package

Database Package: system The Database package supplies services to other classes in the Business-Object package so that they can be stored persistently. In the current version, the Persistent class will store objects of its subclasses to files in the file system.

The diagram in Figure 49 illustrates the package hierarchy of the Optimus Prime application.



Details about the function for each business object are given in Appendix C.1, and their associated UML class diagram in Appendix C.2. Referring to Figure 49, the packages within Optimus Prime divide the application into two functional parts: (i) Database administration: which is controlled by the **system** package and its sub-package **ui**. (ii) Experiment: which is handled by the **engine** and **gui** packages.

The classes with the “ui” suffix represent user interfaces called Panes, which are dialog boxes used for data entry. The classes with the “Data” suffix (e.g. resultsData, materialData, warheadData etc.) they hold information about the object they represent. Their primary role is to read/write/update object data to or from the database or file. One special class, *parameters*, plays a key role in holding all the pre-processing data ready for processing. It is known as a utility class,

and centralises all the information needed to perform the experiment. If any modifications are made to the program, the only the *parameters* class needs to be updated and the associated user interfaces. This centralisation of data minimises the development time when modifications are made. Another utility class, *loadSetup*, holds all the database connection information namely the database driver name, database ODBC source name, username and password. The information is read in from the “setup.ini” file, which is created automatically with default settings if it is not found in the Optimus Prime root directory.

4 Experimental procedure and results

4.1 Experimental criteria and strategy

The application designed has to be tested under the following criteria along with the test plan:

- Experiment 1: Computational performance: Here we test and document Optimus Prime’s computational efficiency. For this purpose we first consider how the computational effort of Optimus Prime scales with small matrix sizes for the fireball and fallout regions, then increment the size for large matrix systems. Both the diffusion and convection-diffusion engines will be tested. This would test the tri-diagonal solver algorithm’s (Crout’s algorithm) efficiency for heavy computational loads. The second test in this criterion will be the image processing performance, i.e. how long the application takes to generate a regional thermal graph displaying fireball and fallout regions (the post-processing graph shown in Figure 46).

All experimental simulations were executed on an 1.8 Ghz Pentium 4 Intel-based workstation with 256 Mb Ram, please note because the workstation is running Windows 2000, not all the RAM is used for the running of the application and application data: a large portion would be dynamically allocated for the running of the operating system.

For a quantitative performance assessment a series of test simulations were carried out on matrix systems of varying size. The chosen matrix sizes for the fireball and fallout regions covered the range from 10,000 to 1,100,000 cells per region. The number of iterations will be kept constant at 20. A graph will show the how the average computation time required for one experimental calculation run scales with matrix system size.

- Experiment 2 : Consistent physical model: The results obtained from the Optimus Prime application have to make physical sense, i.e. accurate when compared to an exploding a real bomb. Optimus Prime allows the physical attributes of the environmental, heat shielding material and nuclear weapon detonation parameters to be modified prior to executing the experiment. Given the permutation of these parameters, one can see that it is very easy to run tests that would bring the project out of scope, so some simplification of the experimental methods are needed here. The two main areas to be tested are:

a). Nuclear weapon physical model: A simple test will be carried out to determine the correlation between the core fireball temperature against the warhead. The yield of the warhead will be incremented starting from the minimum yield to the highest yield and the corresponding core fireball temperature recorded. The thermal response will be measured and analysed to validate the application’s physical model that is concerned with the warhead.

b). Environmental physical model: the atmospheric parameters will be varied from the minimum value to the maximum value, to measure the thermal response from a fixed-yield nuclear bomb to the environment. This test will indicate that the physics engine is mathematically correct and the values are realistic to the conditions set by the user, these are:

i. Convective heat transfer coefficient of air

ii. Specific heat capacity of air

iii. Thermal conductivity of air

4.2 Results

Experiment 1 : Computational Performance

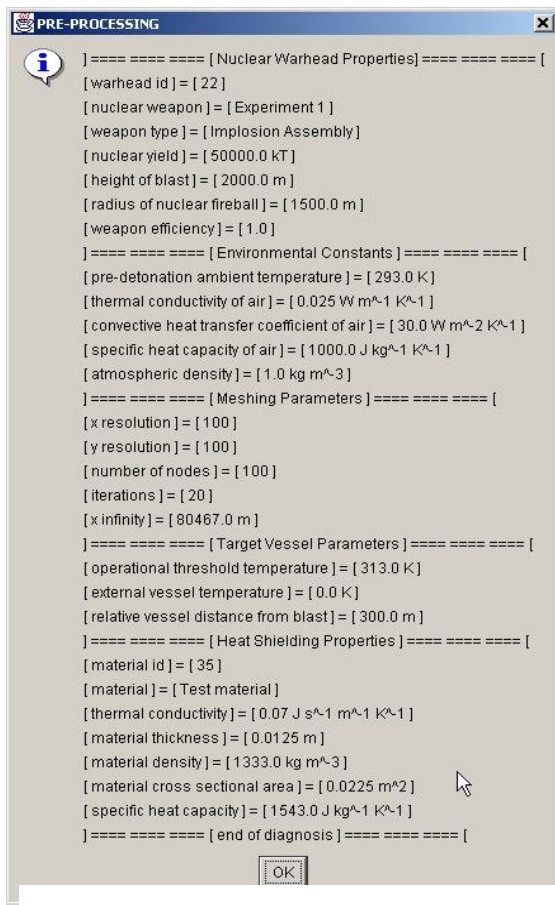
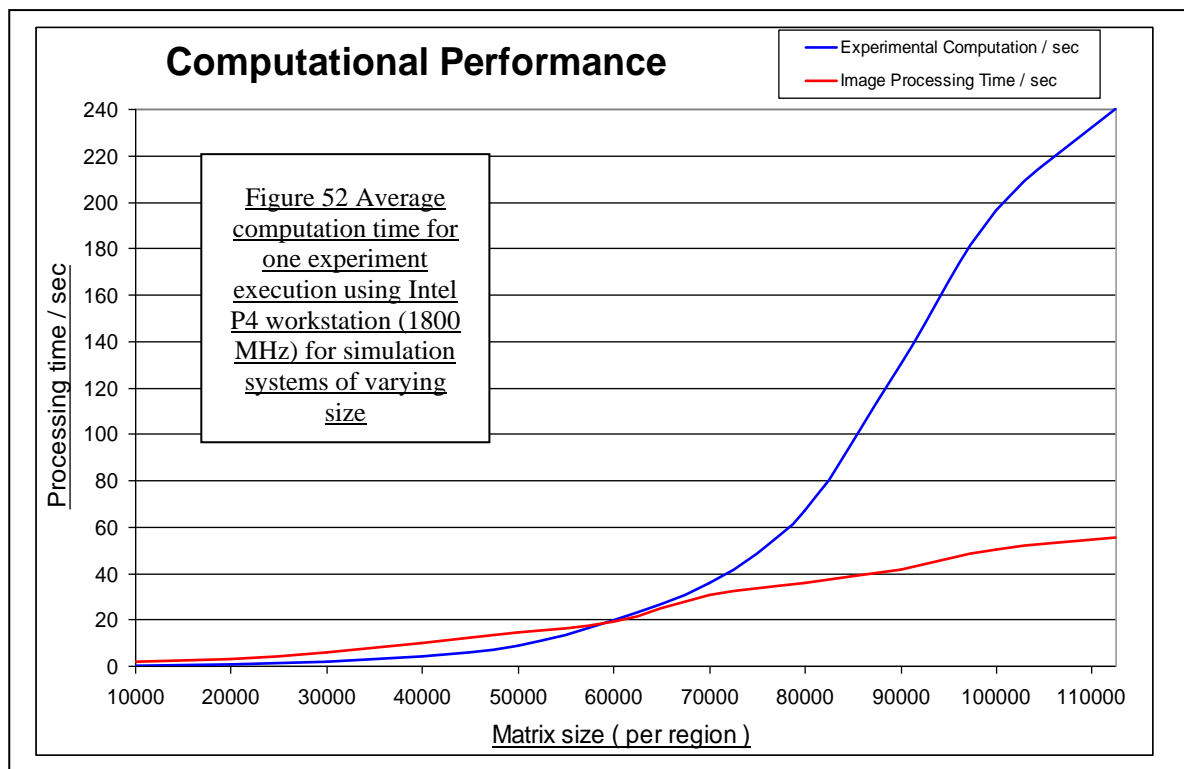


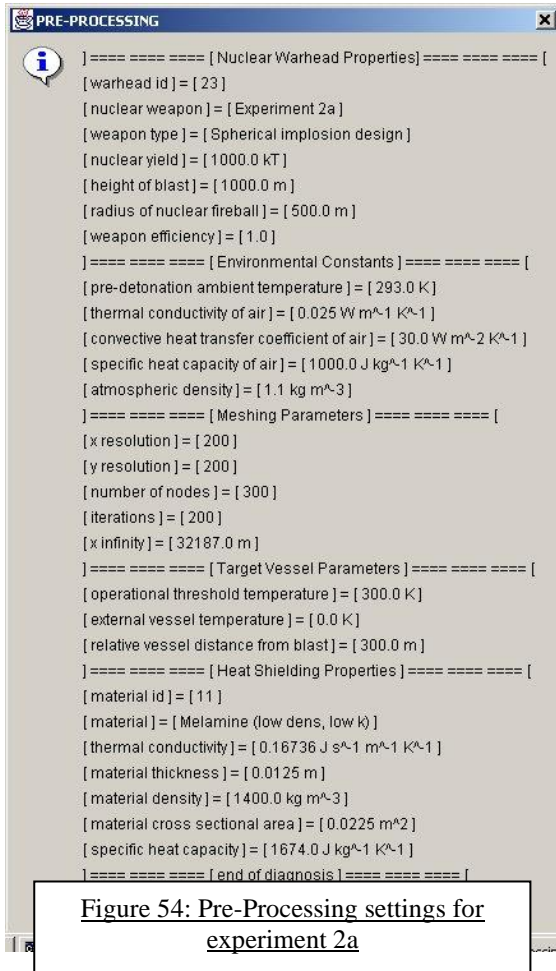
Figure 51 Experiment 1 pre-processing settings

The initial settings for this experiment are giving in Figure 51. Figure 52 the blue line clearly shows that for experimental settings of 20 iterations Optimus Prime achieves n3 order scaling of computational effort with matrix size using the diffusion engine solver. The red line, which represents the post-processing image generation, displays a linear relationship between the computational effort and matrix size. It is evident from these results that the processing time would become appreciably greater for matrix sizes greater than 110,000, it already has taken approximately four minutes to process the matrix size of 110,000 on the Pentium 4 1.8 Ghz workstation! Incidentally, the upper matrix size is set by the limitations on the workstation's memory of 256 MB.

The matrix size can be increased without increasing the physical memory of the system. This is done by modifying the way the matrix is held in memory, i.e. by using sparse matrices to store the data. A matrix is sparse if it contains enough zero entries to be worth taking advantage of them to reduce both the storage and work required in solving a linear system. Ideally, we would like to store and operate on only the nonzero entries of the matrix, but such a policy is not necessarily a clear win in either storage or work. The difficulty is that sparse data structures include more overhead (to store indices as well as numerical values of nonzero matrix entries) than the simple arrays used in Optimus Prime, and arithmetic operations on the data stored in them usually cannot be performed as rapidly either (due to indirect addressing of operands). There are therefore tradeoffs in memory requirements between sparse and dense representations and tradeoffs in performance between the

algorithms that use them. Another disadvantage of using this system is that heavy modifications need to be made to the program infrastructure to accommodate them.

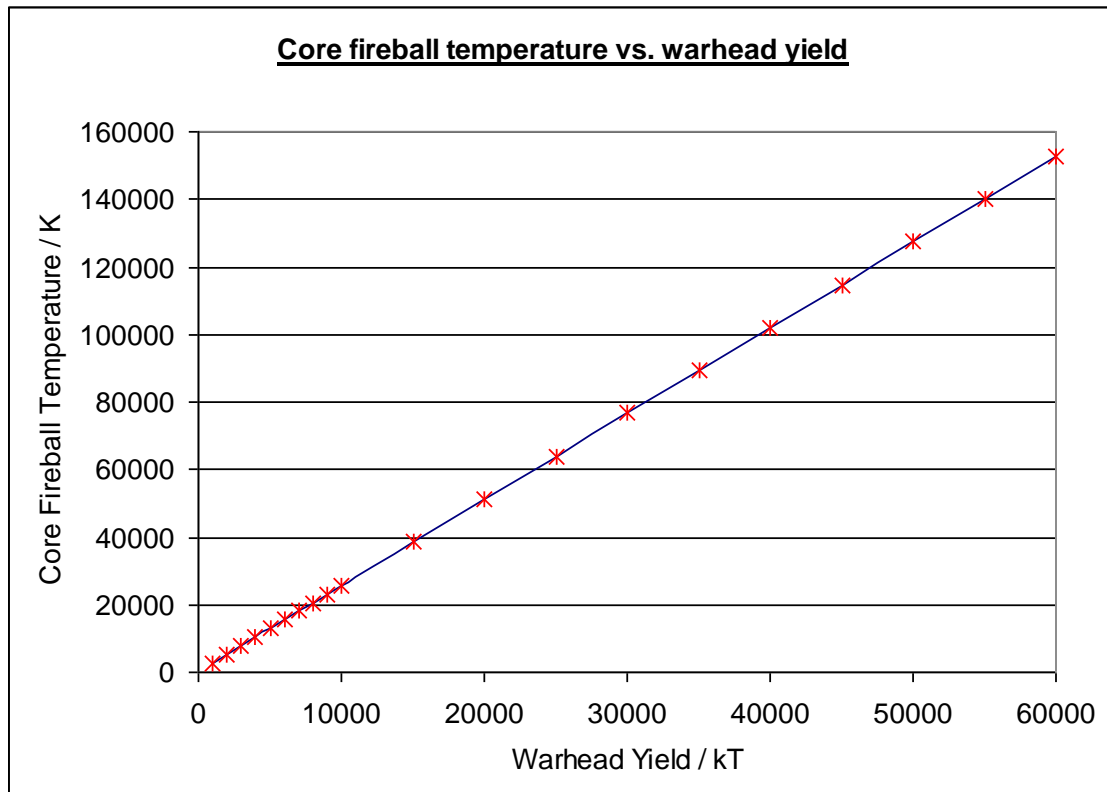




Experiment 2a: Nuclear weapon physical model

The settings for this experiment are giving in Figure 54. The graph in Figure 55 clearly shows a positive linear correlation between the nuclear warhead yield and core fireball temperatures, which is physically correct, as we increase the energy input we get a greater thermal output and hence fireball temperature, but the magnitude of these temperatures are not near those found in a real explosion (that of the order of 10^7 K for a 1 Mt bomb) where the results on graph show an order of 10^5 K maximum. Furthermore, on examination of the thermal output energy graph shown in Figure 56, we can see that the energies given out from the nuclear reaction are extremely high; of the magnitude 10^{16} J, then why are the temperatures of the fireball extremely low? This error can be attributed to the physics model adapted for this application. The thermal physics of any gas changes when very high temperatures are involved. In this case we have adapted an ideal gas model, which is not correct for the temperature range, as super-heated gas usually turns into the fourth state of matter, plasma, which follows an entirely different response pattern to a normal gas. The fireball is assumed to be symmetrical and uniform in temperature at all points within the sphere. This assumption is over-simplified and inaccurate when compared to the real fireball, which has a time-transient varying geometry. The thermal energies obtained from the blast are displayed in Figure 56, they are of, which is correct for a nuclear reaction of this magnitude, this is confirmed by the research shown in Chapter 2.

Figure 55: Graph representing the correlation between the warhead yield and fireball temperature



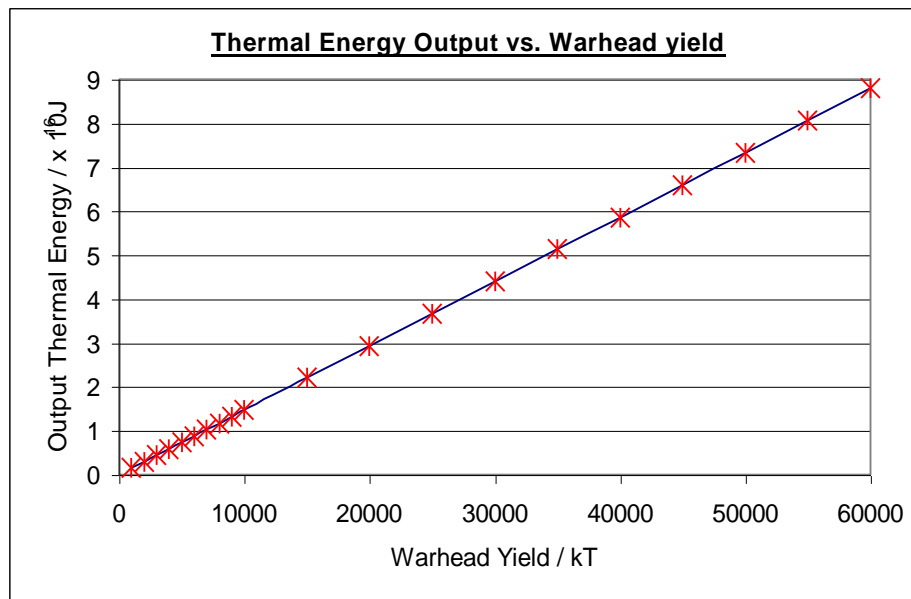
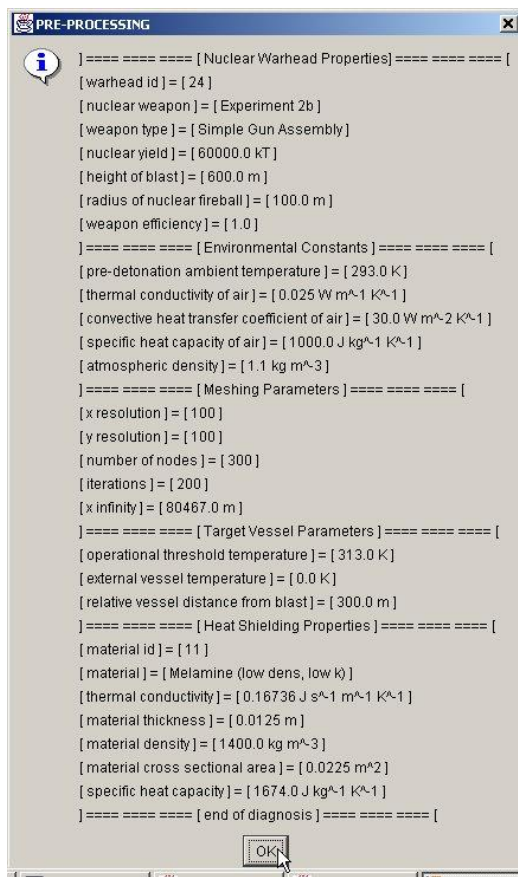


Figure 56: Graph representing the correlation between the maximum thermal output of the bomb and warhead yield

Experiment 2b: Environmental physical model



The initial settings for this experiment are given in Figure 57. The settings given in this figure were kept constant, and only the convective heat transfer coefficient (h) of air varied from a minimum value of $10 \text{ W m}^{-2} \text{ K}^{-1}$ to the maximum value of $50 \text{ W m}^{-2} \text{ K}^{-1}$. The convective heat transfer coefficient is a coefficient that tells us how easily heat can move through a medium via convection. Decreasing this value should display a quicker absorption of heat across each strip in the fallout region; oppositely, increasing this value should display a graph showing a greater displacement of heat across the strip. Theoretically Equation 7 governs this behaviour. The results show that correct physical behaviour is observed in the thermographic profile in Figures 58 and 59 respectively; i.e. the greater the convective heat transfer coefficient, the greater the displacement of heat from the boundary across each strip.

The next physical attribute to be tested was the specific heat capacity of air. The minimum and maximum values were set to $500 \text{ J kg}^{-1} \text{ K}^{-1}$ and $1500 \text{ J kg}^{-1} \text{ K}^{-1}$ respectively, keeping all other variables constant (Figure 57). The specific heat capacity (SHC) is defined as the amount of heat required to change a unit mass of a substance by one degree in temperature. Therefore, a lower SHC value should result in a higher core fireball temperature and a higher SHC should yield a lower core fireball temperature for the same energy weapon. The thermograph in figures 60 (SHC = $500 \text{ J kg}^{-1} \text{ K}^{-1}$) and 61 (SHC = $1500 \text{ J kg}^{-1} \text{ K}^{-1}$) confirms this behaviour: the core fireball temperature in Figure 60 is approximately 29 times higher than the fireball temperature in Figure 61.

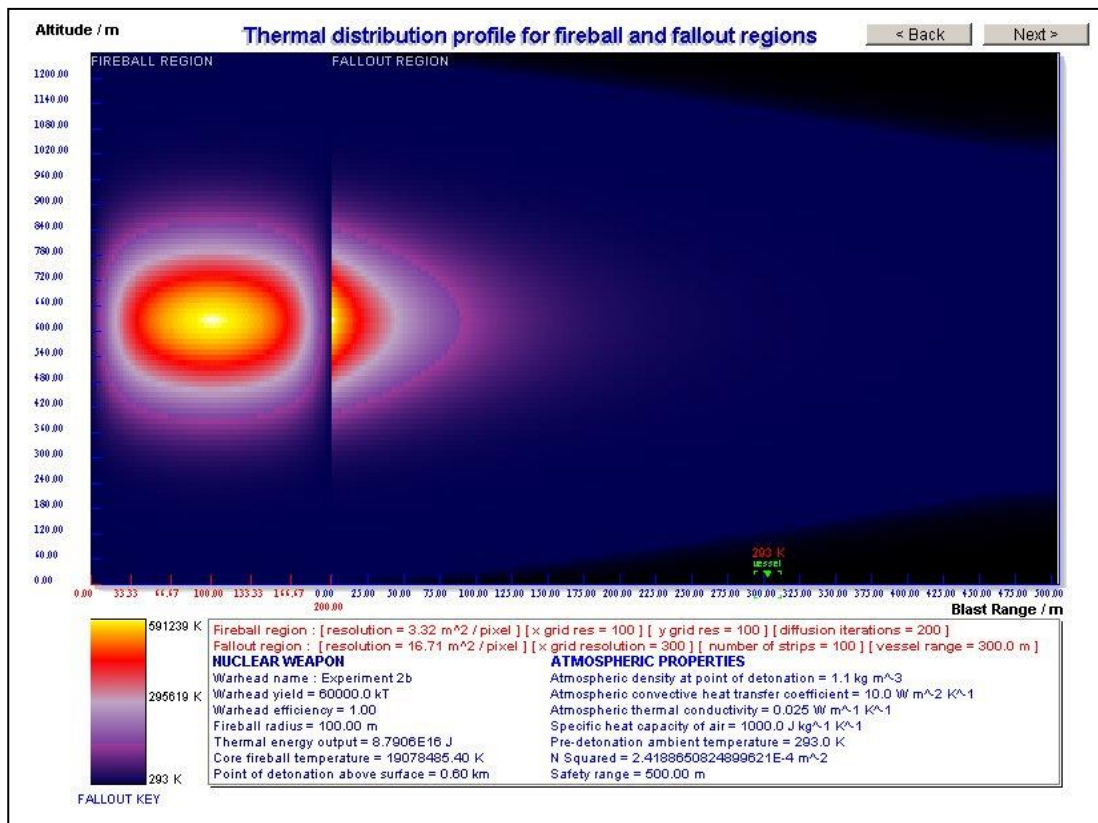


Figure 58: Thermographic profile for convective heat transfer coefficient of 10 W m⁻² K⁻¹

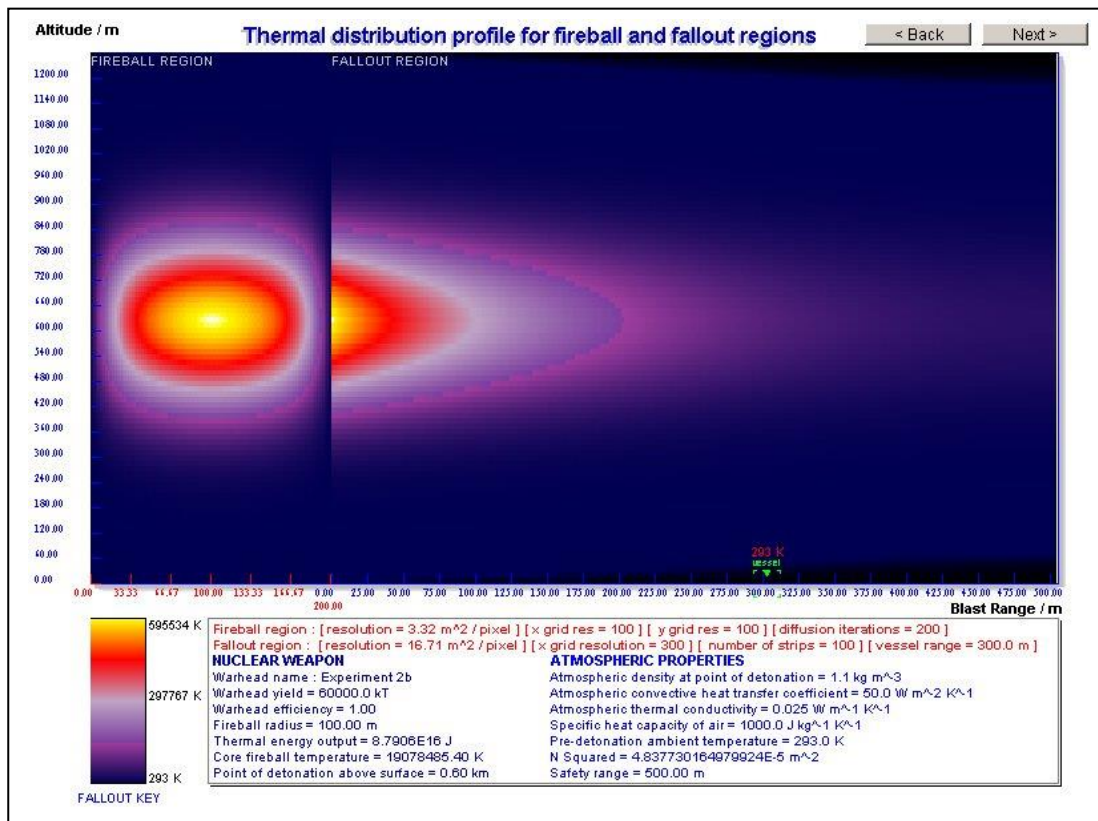


Figure 57: Pre-Processing settings for experiment 2b

convective heat transfer coefficient of 50 W m⁻² K⁻¹

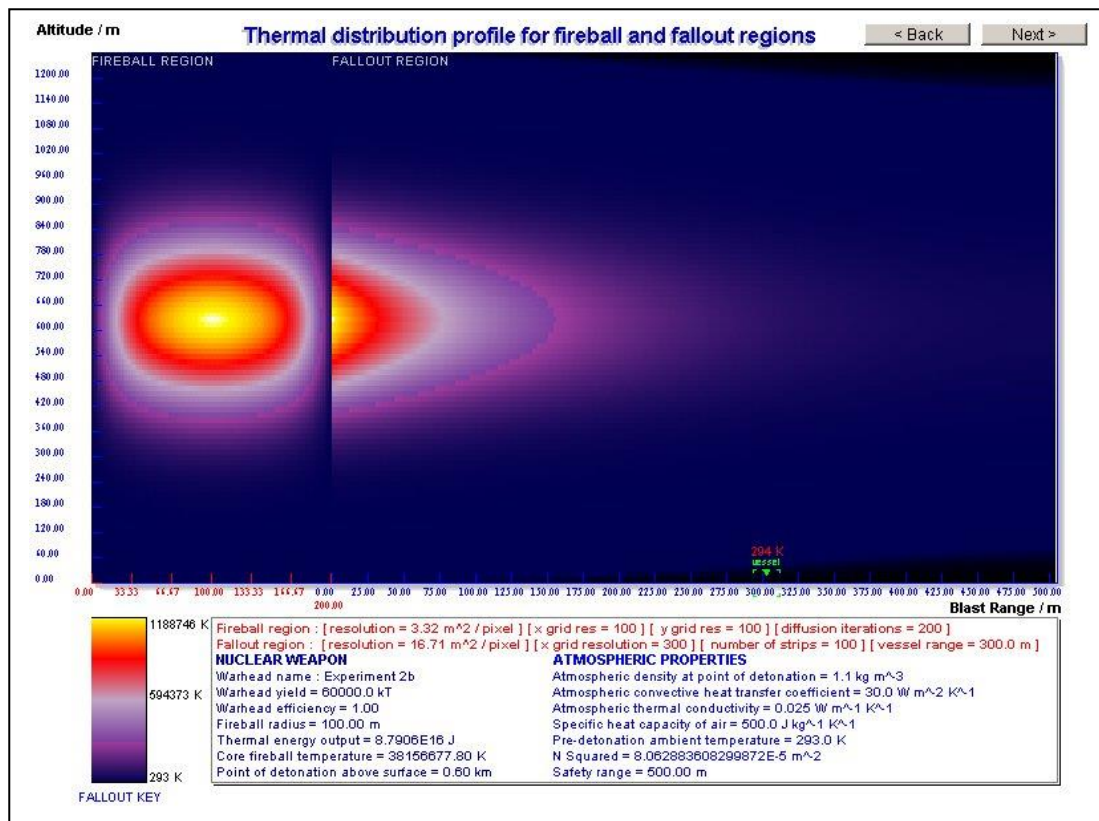


Figure 60: Thermographic profile for specific heat capacity of 500 J kg⁻¹ K⁻¹

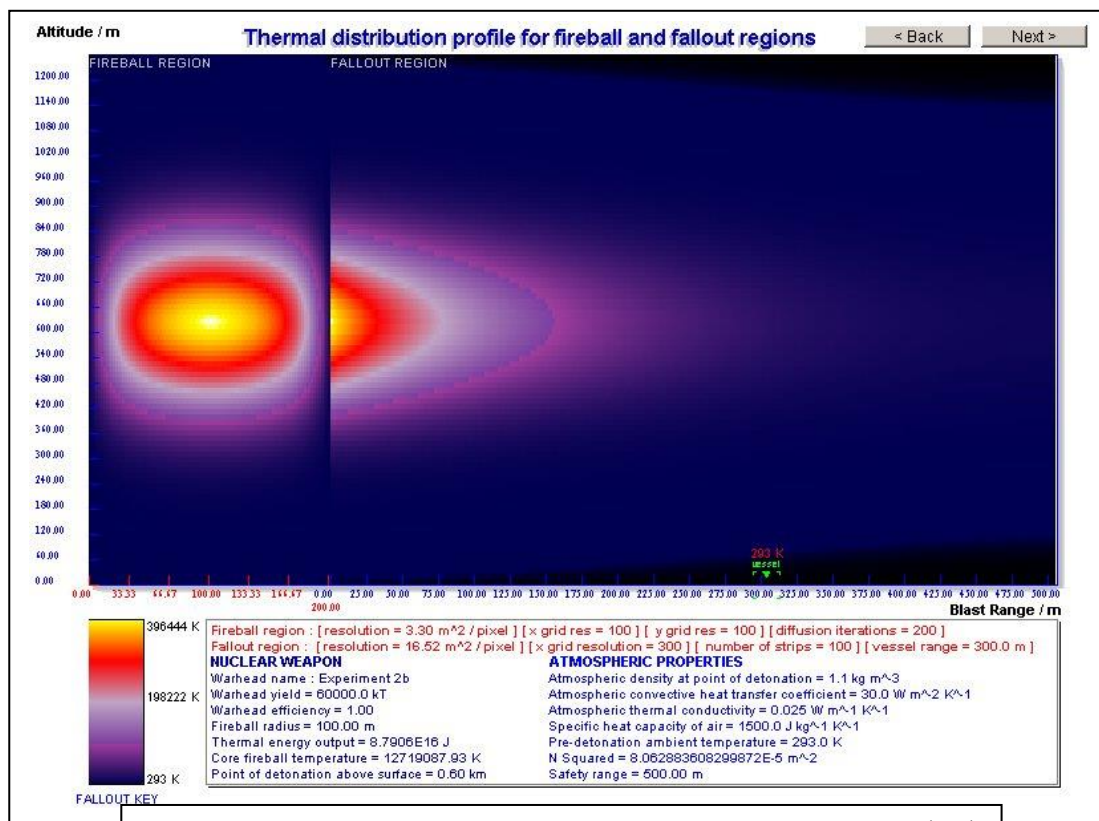


Figure 61: Thermographic profile for specific heat capacity of 1500 J kg⁻¹ K⁻¹

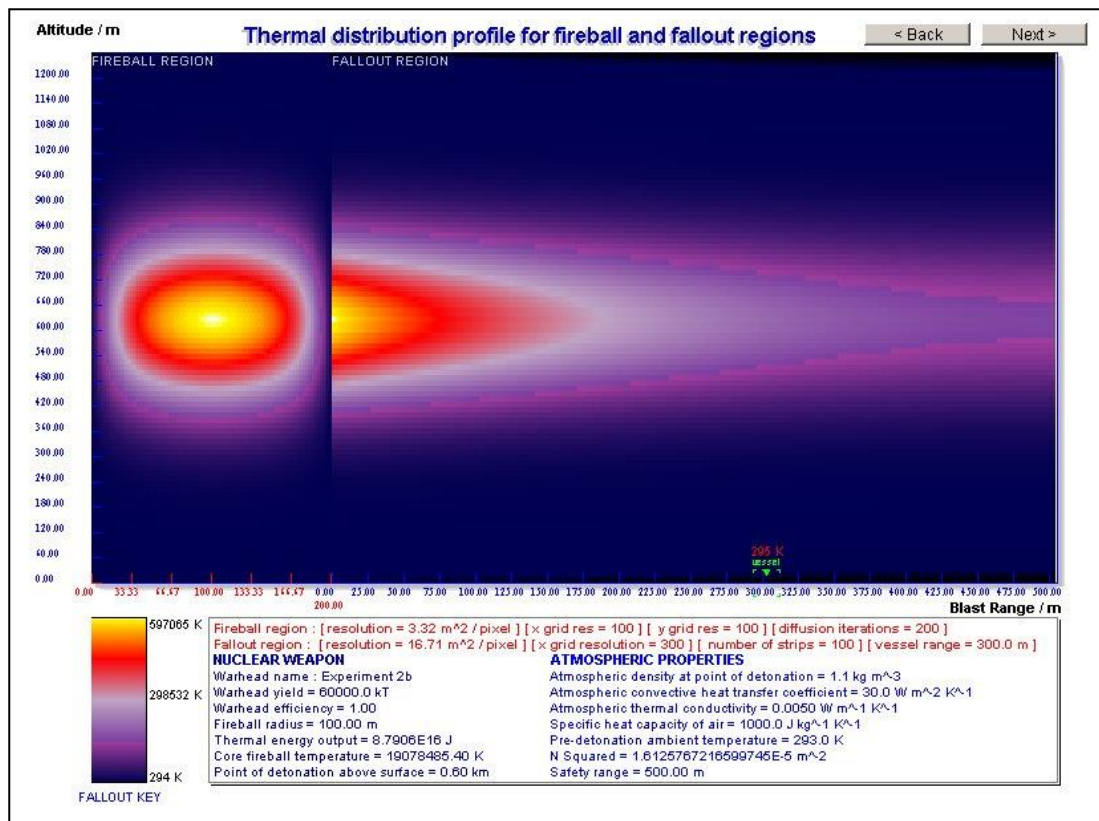


Figure 62: Thermographic profile for atmospheric thermal conductivity of 0.005 W m⁻¹ K⁻¹

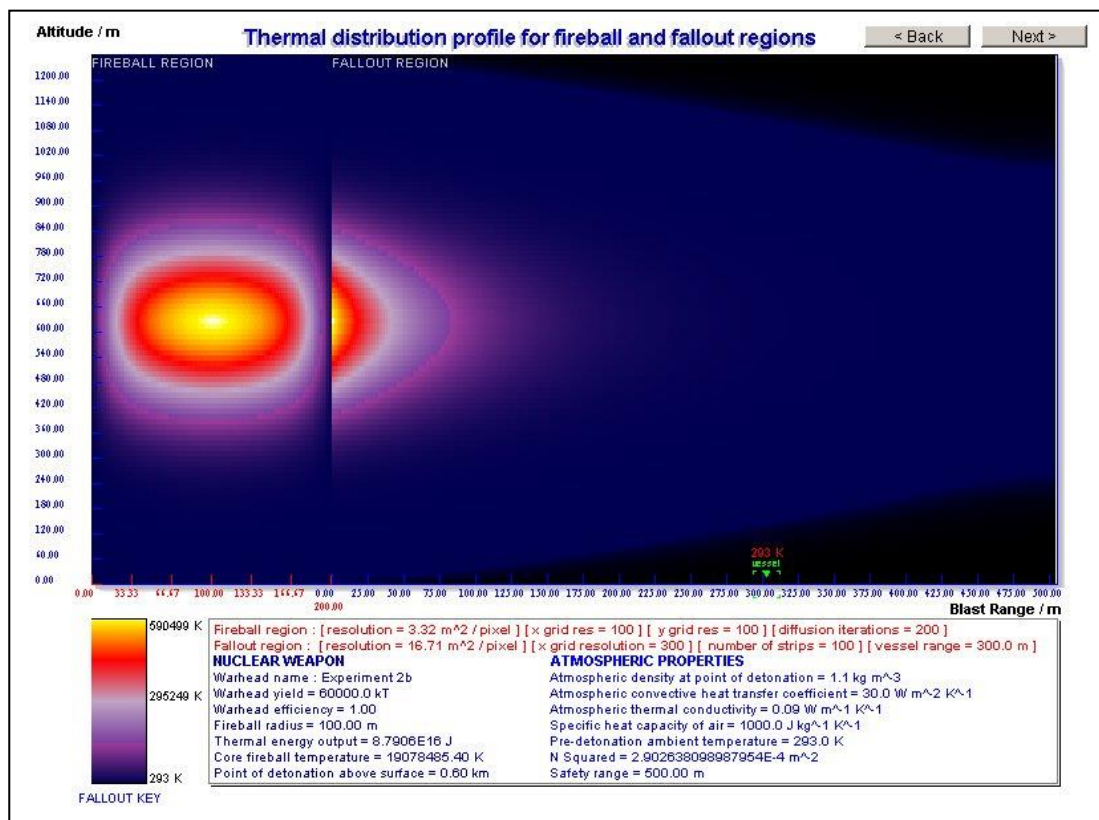


Figure 63: Thermographic profile for atmospheric thermal conductivity of 0.09 W m⁻¹ K⁻¹

Experiment 2b: continued

The last environmental attribute to be tested was the thermal conductivity of air. The values set in this experiment were $0.005 \text{ W m}^{-1} \text{ K}^{-1}$ for the minimum value and $0.09 \text{ W m}^{-1} \text{ K}^{-1}$ for the maximum. Using Fourier's law we can define the thermal conductivity as the rate of heat transfer through a unit thickness of a medium per unit area and per unit temperature difference. A good conductor of heat has a high value of thermal conductivity. Referring to Figure 62, we see that a lower value of thermal conductivity yields better conduction across the fallout region, as opposed to Figure 63, which shows that a higher value for the thermal conductivity displays poorer conduction across the fireball region. This is incorrect! But is not by error in the coding, a deliberate modification was introduced into the code to enhance the thermal effect range across each strip. Inverting the value of n^2 in the strip class was the cause of this fault. Although the thermal conductivity component yields incorrect results with this modification, prior to the modification the results of the experiment were not satisfactory; the diffusion of heat across the strip was very short-ranged; less than one percent of the total range. This was due to the value of n^2 which was dependant on the geometry of the strip in 3 dimensions – due to the enormous size of the strips, the n^2 values were far too great, causing the thermal gradient across the strip to fall to zero quicker. Thus, in order to “adjust” the results, an arbitrary error was thrown into the calculation of the n^2 equation: it was inverted.

5 Conclusion

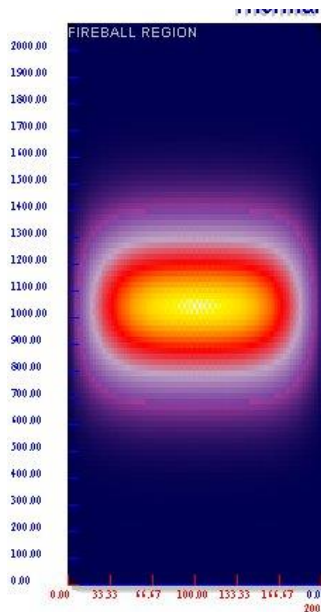


Figure 64: Fireball matrix:
10,000 cell resolution

Although Optimus Prime succeeds in meeting most of the targets set for it in this study, there are some major faults that will be addressed in this conclusion. The first fault lies with the fireball diffusion algorithm (the averaging algorithm in the diffusion class): the physics is inaccurate. The prime purpose of the diffusion algorithm is to create a thermal distribution of the fireball region when the fireball is at its peak temperature. The method the algorithm adapts to achieve this is by averaging the temperatures across the fireball region n times, where n is the number of iterations set by the user. The results from this approach are not realistic, because, changing the number of iterations or the mesh resolution of the region yields a different result each time. For example, examine the fireball

region in Figure 64 - the mesh settings being: $x_{\text{res}} = 100$, $y_{\text{res}} = 100$ and 200 iterations. When the mesh resolution is increased, for a fixed number of iterations, the high fireball temperatures are concentrated around

the fireball region, and dissipate very steeply as we move away from the fireball center, as shown in Figure 65. Clearly, this can affect the outcome of the experiment as a whole; changing the mesh resolution / iterations will have different boundary values, thus affecting the external temperature of the vessel. This calls for complete replacement of the temperature diffuser used, i.e. the averaging algorithm. A more accurate model would be a 2D steady state diffusion algorithm, this model would fit better with the physics of the fireball with its environment. In doing so, increasing the fireball mesh would yield a higher resolution of the fireball region, as opposed to changing the temperature distribution completely.

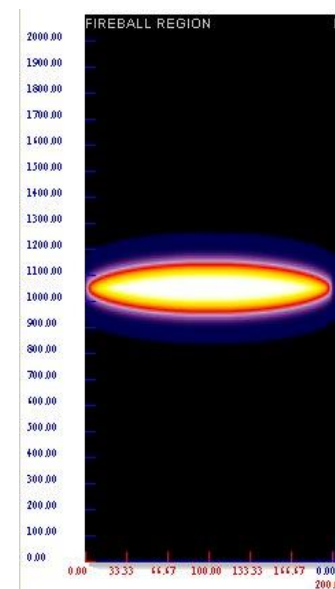


Figure 65: Fireball matrix:
1,000,000 cell resolution

The second issue with Optimus Prime is the quantitative thermographic profile for the fallout region: they are not physically realistic compared to a real life nuclear detonation. Why? Strictly speaking, the assumptions made during the design phase were unrealistic. That is to say the assumptions for the 1-d strip model: heat cannot pass via the upper and lower walls of the strip, i.e. no thermal influence occurs via the neighbouring strips. This would mean that each strip is a closed system, which yields very ordered thermal profile in the fallout region. To overcome this problem, the strip model should be disposed of, and the region should be treated as a complete one-matrix system where a 2-D steady state diffusion algorithm should be employed instead of the 1-D diffusion algorithm. In doing so, this eliminates the

complications of strip geometry, as we would be working in the 2d x-y plane. If this approach is applied, it would nullify the use of the Crout algorithm, as the system of unknown temperatures would not take the form of a tri-diagonal matrix system.

Lastly, the final issue with Optimus Prime is a more trivial one, associated with the user interface. The user is unable to select warheads or materials because the scroll bars are inoperative. The implementation of the scroll bars have been done, but they are not operational, I have spent some time trying to remedy this problem, but the problem lies with the custom Borland (Jbuilder) libraries, which do not allow Swing scroll bar component to be implemented onto a third party Grid (made by Borland). This technicality means that the user can only see a certain amount of items in the dialog box, and is unable to use a scroll bar to view the other items. But if the user uses the keyboard cursor, he/she would be able to select that particular item, but they would not see the item they are selecting until they reach the next screen.

On the other hand, I think that Optimus Prime has accomplished its task well in the areas of user friendly gui, material testing, data management and administration, scientific features and overall functionality. I was extremely pleased with the dynamic array generation and data manipulation abilities of Java, as it saved me much time in coding (no nasty pointers to deal with as in C/C++). Java's performance on the other hand is sluggish due to the security model that checks that the arrays in memory are not in violation of security policies – this is processor intensive and can be extremely slow at times. There are many other features of Optimus Prime that have not been tested, due to scope of this project. For example, the convection-diffusion engine, which calculates the steady state heat flow for fluids of low Reynolds numbers. Note that this is not the correct engine for this type of problem, as the environmental fluid flow model involved in nuclear explosions are turbulent, not steady state. This module was implemented because it contained the same framework code as the diffusion engine, with only a minor modification to the coefficients used to undertake the calculations of each the control volume within the strip.

Overall, this project has shed some light on scientific bespoke software systems; they have their fair share of advantages and disadvantages. The advantages being the scalability of the application as a whole; that is to say its flexibility to adapt for any future improvements, modifications or upgrades, *within the scope of the experiment*. On the other hand, because the system is designed for one particular experiment, it will fail in re-design and development if the nature of the experiment is changed. For example if the post nuclear effects on biological living tissue (as opposed to heat shielding) were to be investigated or the structural integrity of the vessel in the blast path, Optimus Prime would need to accommodate this change; new modules and some major re-engineering to the code will need to be undertaken. This is where off-the-shelf CFD packages will triumph over Optimus Prime. These commercially available packages are adaptable to many different types of problem models, where bespoke systems are problem-specific.

Recommendations: Optimus Prime can be used as a foundation to build a system to test heat shielding materials more accurately, by incorporating modules that would simulate the other two blast effects from the bomb: mechanical and nuclear radiation. Also, phasing the system over to a time-transient model would yield the more scientifically comprehensive results. Even Optimus Prime's simple spatial simulation, modelling just the temperature distribution, requires heavy processing power when the system grows beyond a 60,000 cells per region on a powerful 1.8 Ghz Intel Pentium 4. It would be wise to implement a parallel processing module for the processor intensive algorithms, having java slave clients installed on other machines on the network communicating to the master node using socket calls. Furthermore, to make the application more comprehensive, it would be worth developing on top of the existing system extra modules that allow the user to custom build the warhead prior to the experiment. Currently, the system distinguishes between the warheads by virtue of their explosive yield, implementing this new module would allow the user to construct and uniquely test the output of the warhead in the simulation.

Nuclear testing and safety simulations are an on-going research field, and requires extremely fast computer processing "farms" to be used in parallel, as well as terabytes of memory to generate and simulate environments in 3-D space. These military-grade algorithms are extremely complex in design, and require specialists in more than one field to maintain and operate them.

I believe that the field of scientific computing, especially nuclear bomb test simulations have an important role to play not just in the scientific field, but also in the political arena. Technology of this type is much in demand from governments of countries with nuclear weapons. Whilst doing the research for this project, I have come across many sources that deny involvement in aiding nuclear research for governmental establishments. Most scientists do not believe in aiding the research for weapons of mass destruction, but still continue to contribute passively to the research. The reason being, ironically, that the same governments that harbour these nuclear military technologies fund these very projects that these scientists are working on, and class it under "nuclear safety", which I think is wrong.

6 Acknowledgements

Firstly, I would like to thank God for making this project possible for to accomplish. Secondly I would like to thank Professor K A Pericleous for his help and support throughout this project, as well as my Mother for her support and encouragement.

7 References

Nuclear Radiation Physics by RE Lapp & HL Andrews
Prentice-Hall Ch.19 19.01 Ch.14, 9 10, 11, 18 (1972)

Computational Methods in Partial Differential Equations
A.R.Mitchell John Wiley & Sons Ltd Ch.1 Ch.2 Ch.6 (1969)

Heat Transfer (3rd Ed.) A.J. Chapman, Macmillan Publishing, Ch.1, Ch2, Ch.9, Ch.11 (1974)

Iterative Solution Methods, Owe Axelsson
Cambridge University Press P.24-29, Ch.5, Ch.6 (1996)

Thermal Physics (2nd Ed.), Charles Kittel & Herbert Kroemer, Ch.4, 5, 6
W.H. Freeman and Company (1997)

Irving Kaplan, Nuclear Physics, Brookhaven National Laboratory, Addison-Wesley Publishing Company, Ch.1 - 4, Ch.16 (1958)

An Introduction to Computational Fluid Dynamics - The Finite Volume Method” by "H K Versteeg and W Malalasekera" published by Longman 1998

Java, how to program (3rd Ed.) by Deitel & Deitel published by Prentice Hall 1999

Web references

<http://m707.math.arizona.edu/~restrepo/475A/Notes/sourcea/node67.html> (Crout Factorisation of Tri-diagonal Linear System)

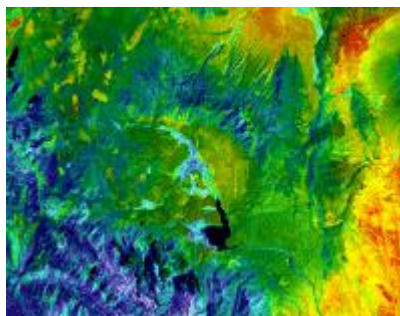
<http://www.math-cs.gordon.edu/courses/cs211/ATMExample/> (Requirement analysis and design tips – UML Professor Russell C. Bjork)

<http://www.umlchina.com/Indepth/DesignJava.htm> (UML design in Java- Hans-Erik Eriksson and Magnus Penker)

<http://www.physics.helsinki.fi/~karhela/mt/node19.htm> (Heat transfer equation for convection - Karhela Tommi)

<http://ie.lbl.gov/education/glossary/glossaryf.htm> (Physics dictionary - [Justin Matis](#))

<http://www.aris.sai.jrc.it/dfc/announce.html> (I got the idea for the colour code from the thermographic image found on this page – refer to the image under the heading “Data Fusion Example Using Landsat 7 ETM+ Imagery” here is a the sample I used to get the RGB values of see below)



<http://search.megaspider.com/b.html?java> (here is the search results for I used to obtain Java tutorials)

<http://www.dartmouth.edu/~krescook/instruct/physguide.shtml> (This is my resource to the scientific community research topics – it is very comprehensive: I used these links to amalgamate the research information found in chapter 2)

<http://quantum.uos.ac.kr/etc/nw.htm> (This is my primary research source for the information about the nuclear weapons and their effects. All the research found in Chapter 2 is based on this resource –Hosted on the University of Seoul website)

<http://www.psr.org/consequences.htm> (Effects of nuclear weapons - I came across this link during my research; very interesting – no information was used from this site, it's here just out of interest)

<http://www.chemie.de/tools/index.php3?language=e> (I used the unit conversion tool on this site for programming purposes)

<http://www.omnis.demon.co.uk/english/homepage.htm?source=index&display=table&format=fixed&cover=beige&screen=%23FFFFFF&text=%23000000&ccover=blue&cscreen=white&ctext=%23000000&cftext=%23ffff00&lang=1®d=false&url=null&link=null&adnum=231&nav=ie5&ver=4.0%20%> (another conversion/utility link I found useful for energy unit conversions whilst I was converting the thermal properties of the heat shielding materials from standard engineering units to S.I. units – why can't engineers speak the same language as us scientists ?)

<http://www.hukseflux.com/thermal%20conductivity/thermal.htm> (information I used for thermal conductivity – needed to find the value of the thermal conductivity of air at normal atmospheric temperature and pressure.)

<http://www.fas.org/irp/threat/mct198-2/p2sec05.pdf> (This file was the closest I was going to get to obtaining military information about the nuclear bomb!)

<http://www.usatoday.com/weather/wstdatmo.htm> (Here I obtained the data for the atmospheric density vs. altitude – used in Optimus Prime's database reference table for determine heat output to the environment at a specific altitude specified by the HOB - USAToday website, source: *Aerodynamics for Naval Aviators*)

http://www.calculator.org/properties/thermal_conductivity_prop (Yet another unit conversion tool used to cross over thermal conductivity values to standard S.I units)

<http://www.nuc.berkeley.edu/thyd/ne161/gregori/section9.html> (Jacobi averaging algorithm tutorial)

<http://www-math.cudenver.edu/~aknyazev/teaching/98/4660/mtp/r2/> (Numerical analysis website used in the programming as a reference)

http://www-jics.cs.utk.edu/PCUE/MOD9_IT/sld002.htm (I went through this tutorial to learn about numerical methods)

http://www.objectsbydesign.com/tools/umltools_byCompany.html (Another programming reference I used for UML)

<http://www.geocities.com/SiliconValley/Vista/2207/sql2.html> (SQL reference for using ODBC databases)

<http://www.developerfusion.com/show/48/2/> (Another good SQL site I used as reference for the “select” statement)

<http://b.sst.ph.ic.ac.uk/angus/Lectures/compphys/node66.html> (Iterative methods reference)

<http://www.krellinst.org/UCES/archive/classes/CNA/dir3.4/uces3.4.html#3.4.2a> (A good site about Block Tridiagonal Matrices and Diffusion in 2D – I used it grasp an understanding of heat diffusion fluids in 2-D and their treatment)

