Turn Up the Heat

Energy Efficiency through Smart Wall Design

New Mexico
Supercomputing Challenge
Final Report
April 2, 2008

Team #61
Los Alamos Middle School

Team Members
Rachel Robey
Jessie Bohn
Teacher
LeAnn Salazar
Mentor
Robert Robey

Table of Contents

Table of Contents i
Table of Figures ii
Executive Summary 1
Introduction 2
- Problem Statement 2
- Objective 2
- Yellow Pine 3
- Why it Works 4
- The Wall 5

Heat Transfer 6
Mathematical Model 8
- Heat Transfer 8
- Night and Day 10

Computational Model 12
- First Order Finite Difference 12
- Boundary conditions 14
- Assumptions 15

Code 16
- Program 16
- Language 17
- Graphics 17

Results 18
Conclusions 20
- Smart Wall Model 20
- Model Capabilities 20

Teamwork 21
Recommendations 22
- Model 22
- Wall Design 22

Appendix 23
- Bibliography 23
- Acknowledgements 23

Table of Figures

Figure 1 Energy usage pie chart 2
Figure 2 Ponderosa - a yellow pine tree 3
Figure 3 Ponderosa pine habitats 3
Figure 4 Ponderosa cones 3
Figure 5 Bark of a yellow pine tree 3
Figure 6 Typical pine habitats 3
Figure 7 Capacitor/resistor 4
Figure 8 Wall design and materials 5
Figure 9 Types of heat transfer 6
Figure 10 Temperatures from www.casaescondida.com/weather.htm 11
Figure 11 An example of a sine wave (max 65, min 10) 11
Figure 12 (Screenshot) The wall by the densities of the materials 12
Figure 13 The dimension of a "cell" 13
Figure 14 State variable and Flux 13
Figure 15 Descriptions of the assumptions 15
Figure 16 Code Flowchart 16
Figure 17 Screenshot 17
Figure 18 Night/Day graphic 17
Figure 19 Five percent resin total energy graph 18
Figure 20 five percent energy difference graph 19
Figure 21 twenty-five percent resin energy difference graph 19
Figure 22 twenty-five percent resin total energy graph 19
Figure 23 Work Breakdown 21

Executive Summary

1) Original wall design doesn't work
2) We DID design a wall that works
3) Our simulation helped us to design effective walls

We came across a wall design that used yellow pine to save energy. The resin melts/freezes at 71°F and reduces the heat flow through the wall. It sounded like it would work, but it didn't. The state change point is too high; it cannot reach that temperature easily, if at all. It doesn't reduce the energy use; however, it may be useful in cooling or if the inside temperature was high enough to let it reach the state change point.

We DID design a wall that works. We lowered the melting/freezing ten degrees, and increased the percentage of resin in the wood. The difference in energy (between regular wood and wood with properties like yellow pine) was only significant if the wood was 10%-25% resin. The state change point can't be too high, or it won't reach that temperature, if it is too low, it won't be able to freeze.

"People seek a challenge just as fire seeks to flame."
~ Chinese proverb.

"The shrewd guess, the fertile hypothesis, the courageous leap to a tentative conclusion—these are the most valuable coin of the thinker at work. But in most schools guessing is heavily penalized and is associated somehow with laziness."
~Jerome S. Bruner (b. 1915), U.S. psychologist

The wall we designed in our program works, but is it buildable? An article we found discussed using salt hydrates, paraffins, and fatty acids for state changing materials in drywall. These materials could also be used in other parts of the wall.

We wrote a program to help us design and test the walls and their efficiency. We use the finite difference method and the computer to calculate the heat transfer quickly and accurately. We can chart the results and compare a wall with wood that didn't have the properties of yellow pine to wood that did. It is also possible to use the program to test the wall in a variety of conditions or expand on it to test different designs of walls.

Introduction

Problem Statement

Energy conservation has become very important, and heating homes is one of many activities that require large amounts of energy and fossil fuels. Conserving heat already in the house, through means such as insulation, cuts down on the amounts and cost of heating. We are researching yellow pine, a natural and renewable resource, which could be used as a building material to prevent heat loss.

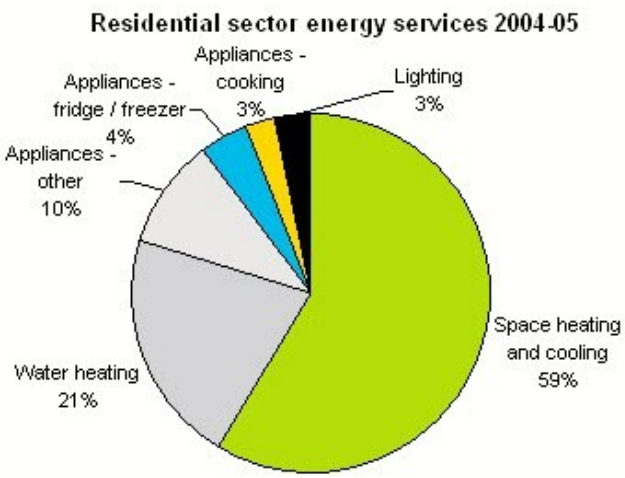


Figure 1 Energy usage pie chart

The resin in yellow pine has a high melting/freezing point, and like all materials, stores energy during a state change. In a wall, it could help to minimize the loss of heat. We will compare the energy that must be added to the inside to maintain a comfortable 70°F temperature. We will run the model for regular wood and that with the properties of yellow pine.

We were initially interested in building because of a house renovation, and happened to come across yellow pine when we were browsing the internet for ideas. We were curious because types of yellow pine grow in our area (such as ponderosa). As we read more about it we discovered that it could be helpful in conserving energy, which was something we were interested in.

Objective

We wanted to create a model to show the wall, time of day, and the amount of heat flow to determine the effectiveness of yellow pine in reducing energy use. For the wall we wanted to have a design that would be accurate and appropriate. The point of the project was to test yellow pine as compared to other building materials, and so our program was to use the wall as a division between the indoors and the outdoors and to show the flow of heat. In our visual we also have a window that shows the temperature of

each cell in the wall model. We used a formula to find a temperature wave that would realistically enhance our program by giving it a night and day as a thermal swing. Night and day is important to the yellow pine because it shows how the resin reacts differently according the outside temperature.

Yellow Pine

Yellow pine includes several different species of trees[7]. Most of these trees are found in southeastern America and include fast growing pine species. Approximately half of all pine species are yellow pines. The most commonly looked for trait of yellow pine is how much resin the pine tree contains.

Ponderosa is one of the most common of the yellow pine trees, but is found in the southwestern interior of North America. They grow around 82 – 98 feet tall and 3 feet wide. Their needles are their leaves, making them conifers. They are often called evergreens because they never completely lose their leaves. The needles grow commonly in groups of three, and are 5-11 inches long, with sharp points at the ends. Cones carry the seeds, and are around 4 inches in length. Cones consist of "scales" with rigid points on them. The cones scales are arranged in a Fibonacci shaped spiral. The bark of mature trees is an orange brown, and has grooves between flaky plates of bark. The bark is tough and protects the trees from common fires. The bark of younger trees is often black.

Figure 4 Ponderosa cones

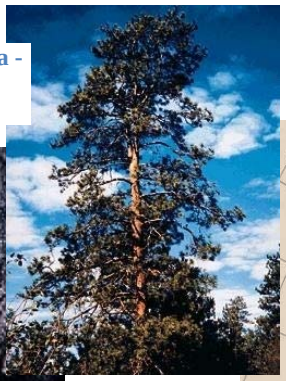


Figure 2 Ponderosa - a yellow pine tree

Figure 5 Bark of a yellow pine tree

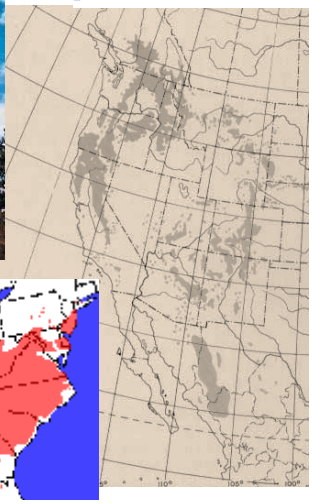


Figure 3 Ponderosa pine habitats

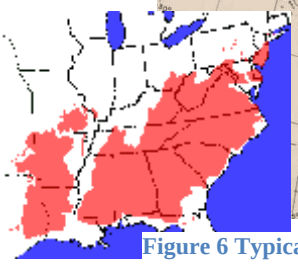


Figure 6 Typical pine habitats

Why it Works

The resin in the wood is what actually stores the energy in yellow pine. It changes state (melt/freeze) at 71°Fahrenheit. When the resin reaches this temperature, the temperature (in the resin) will no longer increase, but the energy will continue to grow. This is because the energy is being used to break the bonds between the molecules, which creates a greater potential energy. However, the kinetic energy, to which temperature is linked, remains steady throughout the state change, only increasing after the state change is complete. In a wall, yellow pine would store much of the heat flowing out, and then emit it when it froze. For example, if it is hot outside, the resin melts, and the heat does not pass through until it has melted. When it becomes cold, it will emit any energy it has stored.

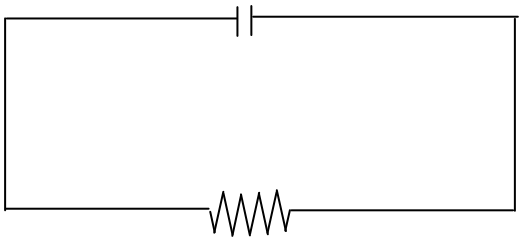


Figure 7 Capacitor/resistor

The materials in a wall act much as the resistors and capacitors in electronics. Something that does not conduct well will act as a resistor by reducing the amount of heat that can flow through. A wall with yellow pine is different because the resin acts as a capacitor by storing energy, while a regular wall (without the occurrence of a state change) contains only materials that act as resistors. Figure 7 shows a resistor and a capacitor.

The Wall

In order to assess the efficiency of yellow pine and other materials, we designed a wall. The current wall has five materials that make up its six parts (segments): stucco, foam sheathing, plywood, fiberglass insulation, sheetrock, and wooden studs. Stucco is used commonly for building where we live. It consists of aggregate (sand, gravel, crushed stone or concrete), a binding material that works as a glue, and water. It is applied to the surface when it is wet and then hardens. It sticks to the next layer in the wall, which is the foam sheathing. The innermost layer is drywall, or sheetrock; this is what is seen from the inside of the house. It is a plaster, which is sometimes mixed with a fiber, and enclosed by heavy paper. Between the foam and sheetrock are wooden studs with fiberglass insulation. The studs are the framework of the wall, and hold up higher floors of the house. The rest of the wall is built off the structure. However, there is a lot of heat loss through the studs because it flows through the wood much more quickly than the insulation. The fiberglass insulation comes in batts (pre-cut) and blankets (roll). It is meant to prevent a large flow of heat, and works best when it's not compressed. We decided on this design because it is relatively common in New Mexico, and it is very similar to the walls we saw constructed in the renovation.

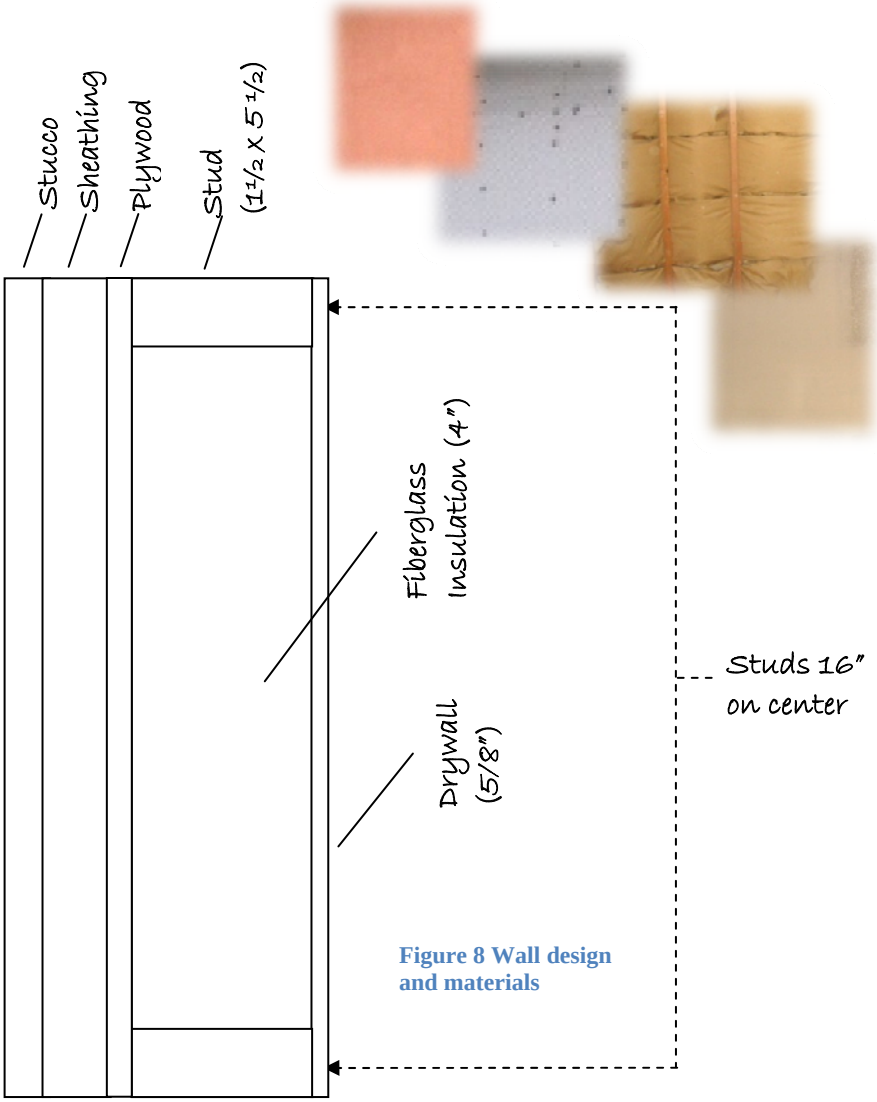


Figure 8 Wall design and materials

Heat Transfer

Some of our initial work was studying the different types of heat transfer and identifying where they would occur in our wall. Heat will always flow from a region of higher temperature to that of lower temperature. We are using all three of the modes of heat transfer in our problem: conduction, convection, and radiation.

Conduction

Conduction is the flow of heat within a substance or between substances in a direct physical link. The molecules pass heat one to another rather than moving. Heat transfer through solid, opaque objects can only occur through conduction. Conduction is found throughout the model: the air, the surfaces of the wall, and the wall. It is also the only heat transfer in the wall.

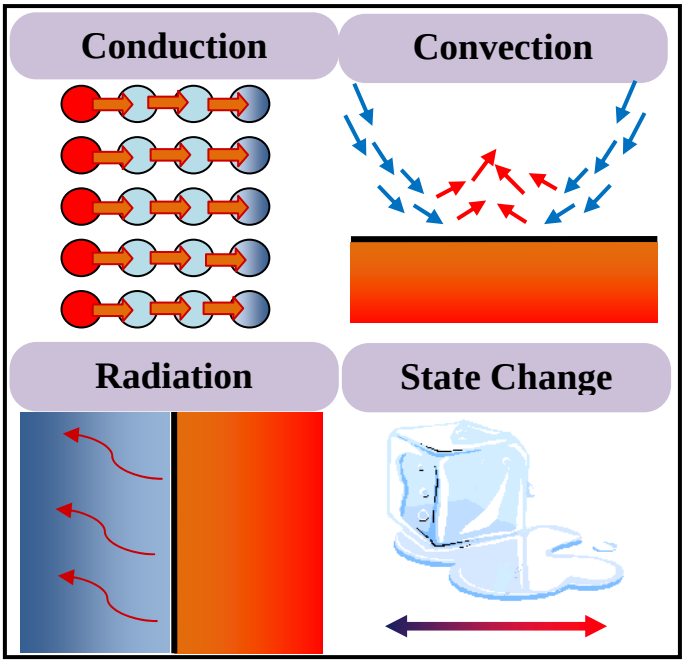


Figure 9 Types of heat transfer

Convection

Convection is the main energy transfer between a solid surface and either a liquid or gas. It is a combination of conduction and storage of energy and heat as well as the motion of the molecules. Through conduction, heat will move from a warm surface to an adjacent fluid or gas. These fluid or gas particles will move to an area of lower temperature because of an increase in temperature. Energy will be passed on to other fluid particles and as they, in turn, move they carry energy and a flow of energy and particles is created. This flow will usually become somewhat circular as some particles lose energy and move back to the heat source. Though it is not strictly a type of heat transfer, it does transport energy relative to a difference in temperature. Convection occurs at the surfaces of the wall. The greater temperature gradient causes there to be more heat flow on the outside surface, and there is very little on the inside.

Radiation

Radiation is the flow of heat from one place to another, without flowing through a material. The heat can flow through even a vacuum, though our project did not require this. There are many other characteristics of radiation that did not apply to our project. Heat travels by waves in radiation, and any object it comes in contact with absorbs its heat. Radiation is found only where the wall radiates heat into the cold air, without moving through the particles.

State Change

A state change is the change of matter from one state to another (solid, liquid, gas). Energy either enters or leaves the material, depending on how the material is changing. The energy will continue to increase (or decrease), but the temperature will remain steady once the state change begins until the process has finished. The energy breaks/loosens the bonds between the molecules, making it less rigid. The kinetic energy does not increase until afterwards, so the temperature does not change either. The state change will occur in the wall using yellow pine, the resin will melt when it reaches 71° and freeze when it reaches the same temperature.

Mathematical Model

Heat Transfer

"Mathematics: silent harmonies. Music: sounding numbers."
~ Mason Cooley (b. 1927)

"I'm not even thinking straight any more. Numbers buzz in my head like wasps.
~ Kurt Neumann (1906–1958), German screenwriter. Kurt Neumann. Dr. Lisa Van Horn (Osa Massen), Rocketship X-M, after hours of calculations (1950)

The types of heat are represented by two equations. One is the flow of heat by conduction, and the second is conduction and radiation, which are combined.

The rate of heat transfer by conduction, q_k, is as follows.

$$q_k = -kA\frac{\Delta T}{\Delta x}$$

- k, the thermal conductivity of the material
- A, the area through which heat is flowing, measured perpendicular to the heat flow direction
- $\Delta T/\Delta x$, the difference in temperature over the distance with respect to the direction of heat flow

The rate of heat transfer by convection (combined with radiation), q_c, in Btu (British thermal unit) per hour is the product three different values.

$$q_c = \bar{h}_c A \Delta T$$

- h_c, the convective heat transfer coefficient, the average unit of thermal convective conductance
- A, the heat transfer area, measured perpendicular to the heat flow
- ΔT, difference between the surface temperature, and a temperature of the fluid/gas

The equation of state shows the relationship between energy and temperature. Although it is not completely linear, it becomes linear after a point. Since we are only working in the area after that point, we can treat the rest as linear.

$$E = c_v T$$

- c_v stands for the specific heat of the material, which is the amount of heat required to raise a quantity of weight one degree
- T is the temperature of the material at that point

When the resin melts, the temperature will not rise. It was necessary to find how much energy was needed before the temperature again begins to increase, in order to mimic the change of state mathematically.

We were able to find the value for the vaporization of several types of resins, though not specifically that of a yellow pine. We began by converting the value into the correct units. We are working in Btu/lb rather than cal/mole. We converted by multiplying by a ratio of the two units that is also equal to one. For example, we multiplied by $\frac{1000\ cal}{kcal}$ and it canceled the calories and kilocalories. Since it is also amounts to one, it does not change the answer when you multiply.

$$E_{vap} = \frac{13.44\ kcal}{mole} = \left(\frac{13.44\ \cancel{kcal}}{\cancel{mole}}\right)\frac{\cancel{mole}}{154\cancel{gm}}\left(\frac{1.8\ Btu/lb}{\cancel{cal}/\cancel{gm}}\right)\left(\frac{1000\ \cancel{cal}}{\cancel{kcal}}\right) = \frac{24192\ Btu}{154\ lb}$$

This is the value for vaporization; the resin will be melting, and the energy for fusion (melting) is about fifteen percent of that of vaporization. Also, about five percent of wood is resin. Find the fifteen percent of the value above (times 0.15), and then five percent (times 0.5).

$$E_{fusion} \cong 15\%\ of\ E_{vaporization} \qquad \sim 5\%\ of\ wood\ is\ resin$$

$$E_{fu} = (0.15)(0.05)E_{vap} = \frac{1.175 Btu}{lb_{wood}}$$

The final result is 1.175 Btu per pound of wood. This is the amount of energy it takes to melt the resin in every pound of wood. The value is approximate, the source didn't have specifically yellow pine; however, it will still show the effectiveness of materials (specifically resin) that conduct a state change in the wall, to preserve heat. We can change the percentage of resin in the wood by using a different percentage in decimal form in the place of 0.05.

Night and Day

We decided to create a thermal swing that resembles a night and day to more accurately test the wall. This will allow us the resin to store energy during the "day" in order to emit it when it is cold. We composed a sinusoidal wave for a minimum and maximum temperature.

A general sine wave is:

$$y(t) = A \times sin(\omega t + \theta)$$

- A is the amplitude, or the distance from the center
- ω is the angular frequency, which is the radians *per unit of time* and is equal to $\frac{2\pi}{T}$
- θ is the phase, or the placement of the wave

The placement on the y axis, using the average (mean)

$$\frac{Tmax + Tmin}{2}$$

The amplitude is shown as

$$\frac{Tmax - Tmin}{2}$$

Angular frequency is two pi over time. Since we measure the day in minutes, and there are a total of 1440 minutes in a day

$$\frac{2\pi}{1440} = \frac{2\pi/2}{1440/2} = \frac{\pi}{720}$$

Phase is set using the current time and the time of a peak

$$t + t_{offset}$$

These terms are combined to create a formula which is used to find a temperature at any time for a given daily temperature maximum and minimum.

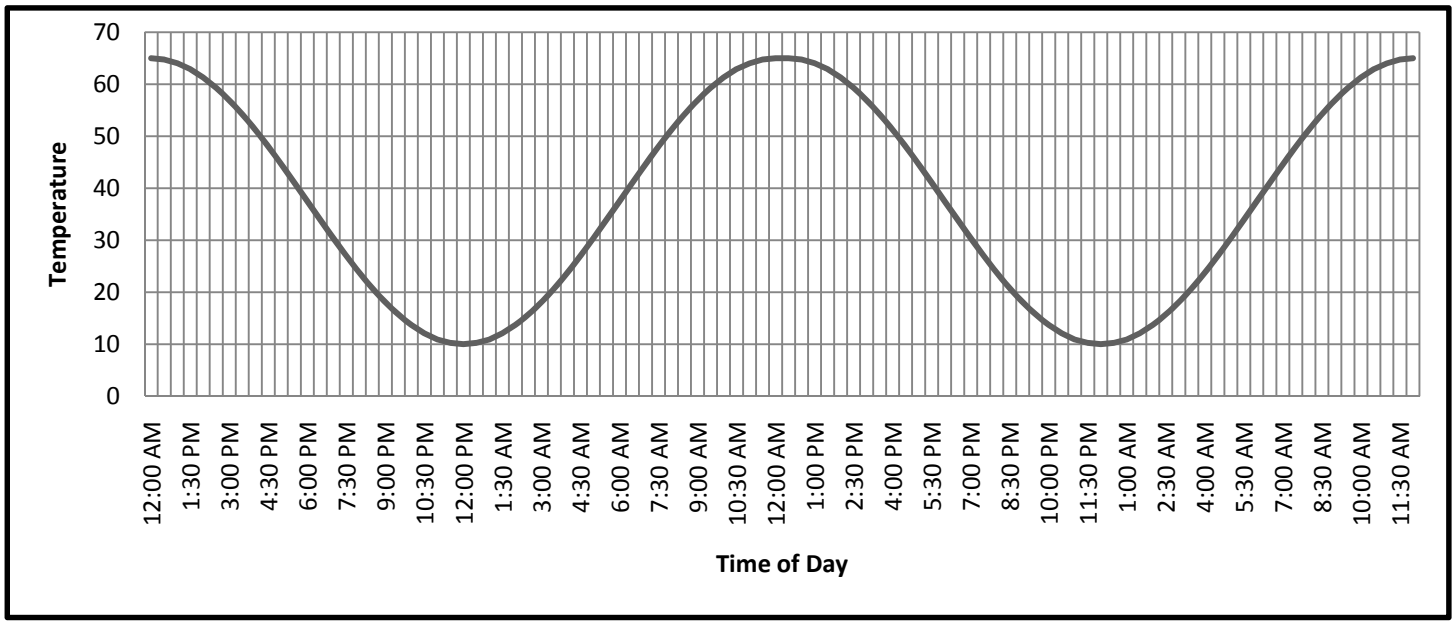
$$T = \frac{Tmax + Tmin}{2} + \frac{Tmax - Tmin}{2} sin\left(\left(t + t_{offset}\right)\frac{\pi}{720}\right)$$

- T is the temperature
- Tmax, Tmin are the maximum and minimum temperatures
- t is the time
- t_{offset} is the shift of the peak, in this case to make it midday, *midnight*
- $\frac{\pi}{720}$ is the angular frequency and sets how often a peak occurs

Month	High	Low
January	42°F	19°F
February	48°F	23°F
March	55°F	25°F
April	64°F	34°F
May	73°F	43°F
June	84°F	52°F
July	87°F	57°F
August	84°F	55°F
September	80°F	49°F
October	68°F	38°F
November	52°F	25°F
December	45°F	19°F

Figure 10 Temperatures from www.casaescondida.com/weather.htm

Figure 11 An example of a sine wave (max 65, min 10)



Computational Model

First Order Finite Difference

For the computational model we broke our wall into cells of equal size. We work with the temperature/energy at the center of each cell. Since we choose to make each cell one inch by one inch, some of the actual widths of the materials had to be approximated. The smaller the cells are, the more points there are at which you know the conditions. Make note that the wall is a horizontal cross-section.

Computers are good at swift, accurate computation and at storing great masses of information. The brain, on the other hand, is not as efficient a number cruncher and its memory is often highly fallible; a basic inexactness is built into its design. The brain's strong point is its flexibility. It is unsurpassed at making shrewd guesses and at grasping the total meaning of information presented to it."
~ Jeremy Campbell (b. 1931)

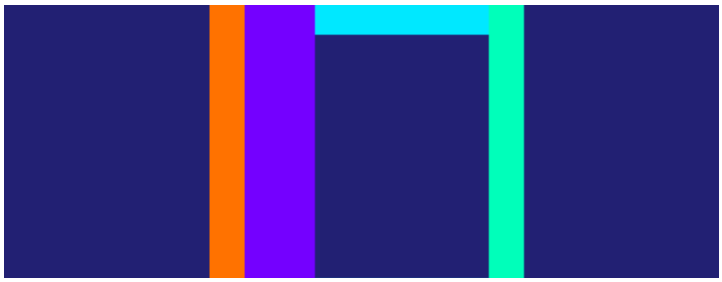


Figure 12 (Screenshot) The wall by the densities of the materials

We used a first order finite difference method to calculate the energy in the cells. Our project uses boundary problems, finding interior conditions from the conditions at the boundary, which is commonly solved by finite difference. In this case, we are using central difference.

Finite difference is used to find the value in the center of each cell in the next iteration, after heat transfer has been calculated. We use information in the current time step to recalculate the energy. This is shown by:

"From then on, when anything went wrong with a computer, we said it had bugs in it."
~ Grace Murray Hopper, on the removal of a 2-inch-long moth from an experimental computer at Harvard in 1945, quoted in Time 16 Apr 84

$$U_{i,j}^{n+1} = U_{i,j}^{n} + \frac{\Delta t}{\Delta x}\left(F_{i+\frac{1}{2},j}^{n} - F_{i-\frac{1}{2},j}^{n}\right) + \frac{\Delta t}{\Delta y}\left(F_{i,j+\frac{1}{2}}^{n} - F_{i,j-\frac{1}{2}}^{n}\right)$$

In $U_{i,j}^{n+1}$, U is the state variable with time in the superscript and spatial in the subscript. This term describes the conditions in the next time step $(n+1)$ at a place (i,j). $U_{i,j}^{n}$ stands for the current conditions at a point in space. $\frac{\Delta t}{\Delta x}F$ is

$$F\left(\frac{Btu}{hr - ft^2}\right)\Delta t(hr)\frac{\Delta y \Delta z (ft^2)}{\Delta x \Delta y \Delta z (ft^3)} = F\left(\frac{Btu}{\cancel{hr} - \cancel{ft^2}}\right)\Delta t(\cancel{hr})\frac{\cancel{\Delta y}\cancel{\Delta z}\,\cancel{ft^2}}{\Delta x \cancel{\Delta y}\cancel{\Delta z}\,ft^3}$$

simplified. The term $\frac{\Delta t}{\Delta x}\left(F_{i+\frac{1}{2},j}^{n} - F_{i-\frac{1}{2},j}^{n}\right) + \frac{\Delta t}{\Delta y}\left(F_{i,j+\frac{1}{2}}^{n} - F_{i,j-\frac{1}{2}}^{n}\right)$ is the flux at each border in the cell, a half (step) taking you from the center of the cell to the boundary. Energy will either be added or subtracted from the state variable value.

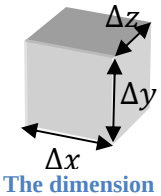


Figure 13 The dimension of a "cell"

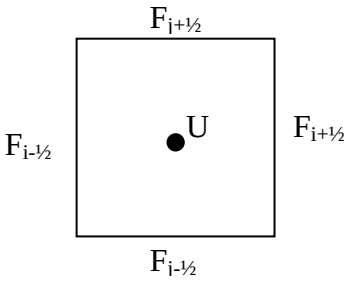


Figure 14 State variable and Flux

We are using *first order* finite difference, which means we find the flux once every time step. This is a linear interpolation between the two points and a flux in the center. Second order, on the other hand, finds the flux halfway through the time step and then again at the end. When we calculate the flux, and we find the flux at each boundary, rather than go cell by cell and find the difference at each of its boundaries. If we were to go cell by cell, we would compute the flux at the same boundary twice.

Our problem requires only one conservation law, that of energy. There are no differences in density, mass, or momentum, only energy. The conservation of energy is shown as $E_t + q_{x,y} = 0$ or, in matrix form,$[E]_t + [q]_{x,y} = 0$. This is a combination of a state variable and a flux term, which are the differences at the boundaries. These are commonly referred to as “U” and “F” in numerical methods. E_t is a shortened form of $\frac{\partial E}{\partial t}$,

which stands for energy relative to time. $q_{x,y}$ is the flux term. This expands to $q_{k,x} + q_{c,x} + q_{k,y} + q_{c,y}$ which are the equations for heat transfer (q_k being conduction and q_c being a combination of convection and radiation) in the x and y directions. This is a two dimensional model, which works well for the problem we are modeling.

Boundary conditions

We needed to be able to calculate the energy/heat that needed to be added to the "house" to keep it at a comfortable 70°F temperature. We maintained this temperature at the inside boundary, but we had to find the temperature it would be in order to track the energy that must be added. We set the outside boundaries of the ghost cells to zero so as to eliminate the loss or gain of heat. We knew the how much energy needed to be added to maintain the seventy degree temperature through subtraction.

The outside boundary follows the sinusoidal wave that we created. The minimum and maximum are variables to make it easy to test the wall in different temperatures.

Assumptions

We made several assumptions in our model which occasionally cause it to differ from real life. The temperature swing (sinusoidal wave) is a good approximation, but a real temperature history would be more accurate. Also, the radiation and convection are combined to calculate the heat loss at the wall surfaces. This does not show the larger heat loss that is experienced on windy days, and there is no difference in the loss/gain of heat through radiation on clear days as compared to cloudy days. The model is two dimensional and does not show some things are only possible with three dimensions, but this is not as important in our model as in others. Last of all, it was necessary to change the melting/freezing point to 61° since the highest temperature in the model is 70° and it is impossible that the resin can reach its 71° state change point. The biggest inaccuracies of the model are because we were unable to find exact numbers for things such as the energy needed for fusion.

Assumption/Limitation	Description
Temperature swing – sinusoidal wave	Good approximation, not completely accurate
Radiation and convection combined	Does not show different heat loss/gain for different weather
2-D	Three dimensions are necessary for some actions/properties; however, it is not as important in our model as in others
Melting/Freezing at 61°	Because the highest temperature in the model is 70°, it is impossible that the resin can reach the 71° melt/freeze point
Cells are one inch by one inch	A higher resolution would give more accurate results

Figure 15 Descriptions of the assumptions

Code

Program

We began with an example from the supercomputing kickoff of a shallow water model, and modified it for heat transfer. Since we were learning the language as we did our project, it was helpful to have the structure of the code. We began by setting the wall into the program, and then we added the equations. Once we had it running, we added some of our own touches to the graphics (see below). We continued to improve the program; one of the biggest changes was putting in the y direction. It was only later that we wrote code to give the wood properties of the yellow pine. We also inserted a sheet of plywood that was not in the original wall. Some of the initial results didn't make sense, so we went through the program and equations a few times to get it to work properly.

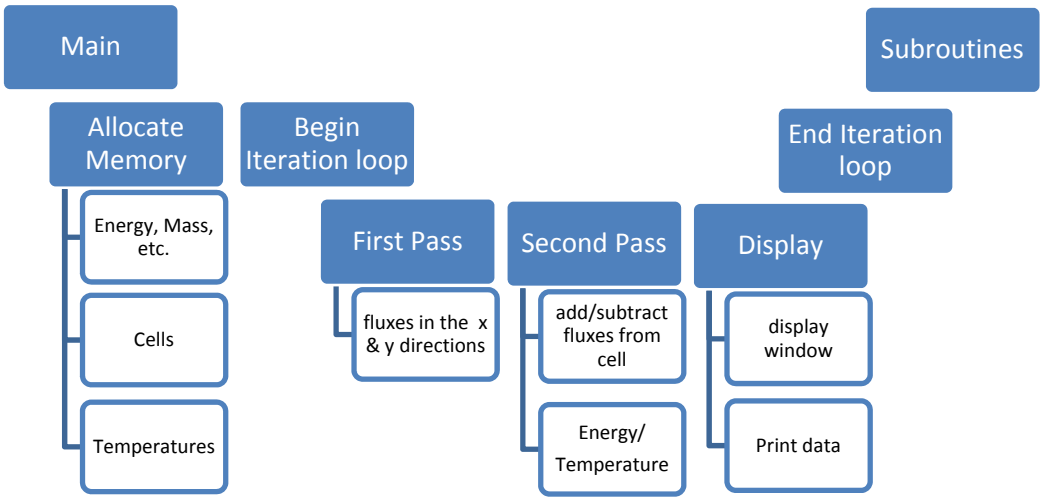


Figure 16 Code Flowchart

Language

We are working in C, and using the libraries MPI (Message Passing Interface) and MPE (Multi-Processing Environment). We have the MPI calls in the code, but we have not run the program on multiple processors (parallel) or debugged it. We use the MPE library for the graphics.

Graphics

Our graphics window is divided into top and bottom halves. At the bottom, it displays what is being shown, the iteration number, and the run time. The top shows the cell characteristics. When the program is first started, it shows the wall by the densities of the materials. After a set wait time, it begins to run, and shows the temperature (Fahrenheit) of each cell (red =hotter <--->blue =colder). The bottom half shows a line that cuts through the wall and displays the temperature at each of the cells it cuts through. The two black lines show the inside and outside edges of the wall. There is an approximate scale on the left-hand side, which will give rough estimate of the temperatures shown by the red lines. We also put in a display of the time (relative to the run time) and the current outside temperature (according to the sinusoidal wave). A sun is shown in daytime between 6 AM and 6PM. A black moon is shown during night hours.

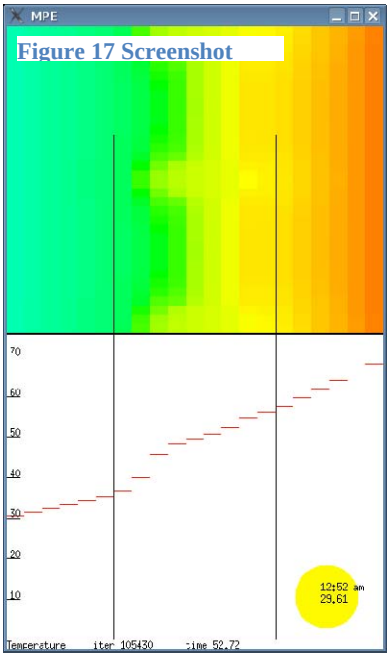
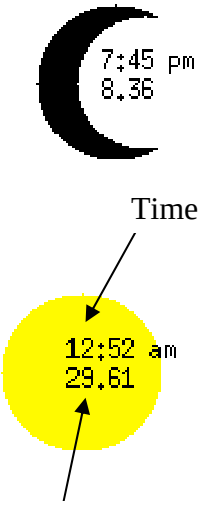


Figure 17 Screenshot

Figure 18 Night/Day graphic



Results

"The true finish is the work of time, and the use to which a thing is put. The elements are still polishing the pyramids."
~ Henry David Thoreau (1817–1862)

In order to chart our results, we read data out of the program to a CSV file. Since we estimated that about 5% of wood is naturally resin, we ran our code with that value first. The first times we tested a percentage of resin, we ran one million iterations. Occasionally we would run it again with ten million iterations. The fewer iterations, the less time it takes for the program to run; however, a longer run time shows the long term effects (including night and day) of the material. We recorded both the total energy that must be added and an energy difference. The total energy is the amount of energy that must be added to the inside border to maintain a seventy degree temperature, and continues to add onto itself. The energy difference is the energy needed for every iteration (although we only record every thousandth). The night and day create a wave in the energy. The house needs to be heated less in the day and more at night.

With a five percent resin there wasn't much of a difference between the yellow pine wood and the regular wood. It is just below the wood in the total energy graph (left). The wave in the total energy is more subtle than that in the energy difference because the change is a smaller part of what is being shown and impacts it less.

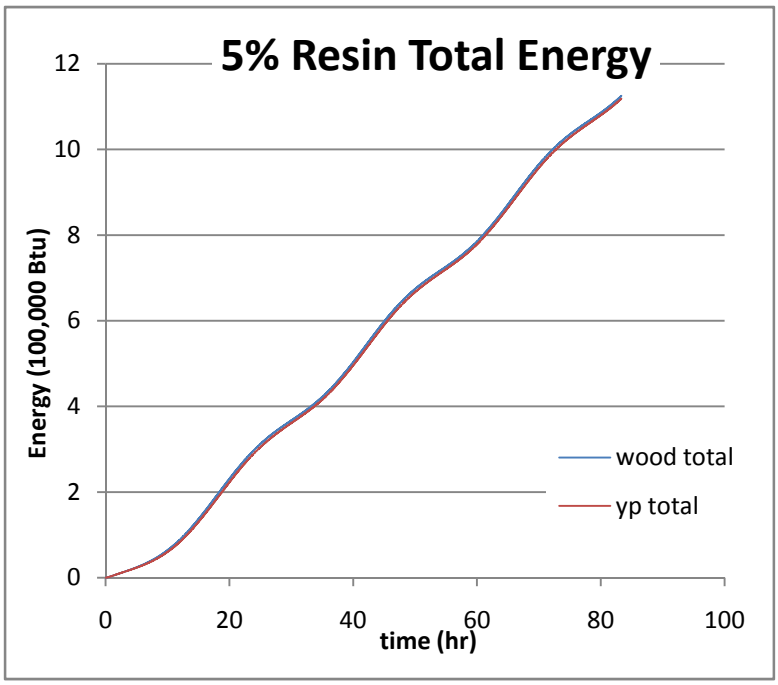


Figure 19 Five percent resin total energy graph

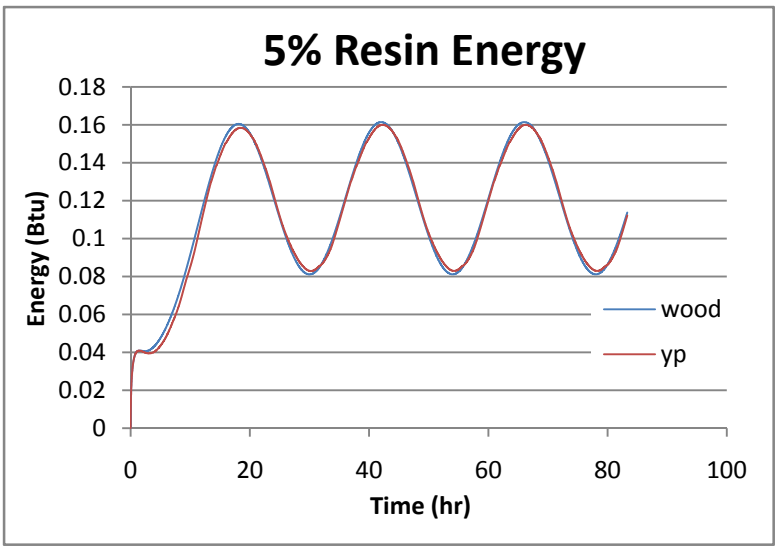


Figure 20 five percent energy difference graph

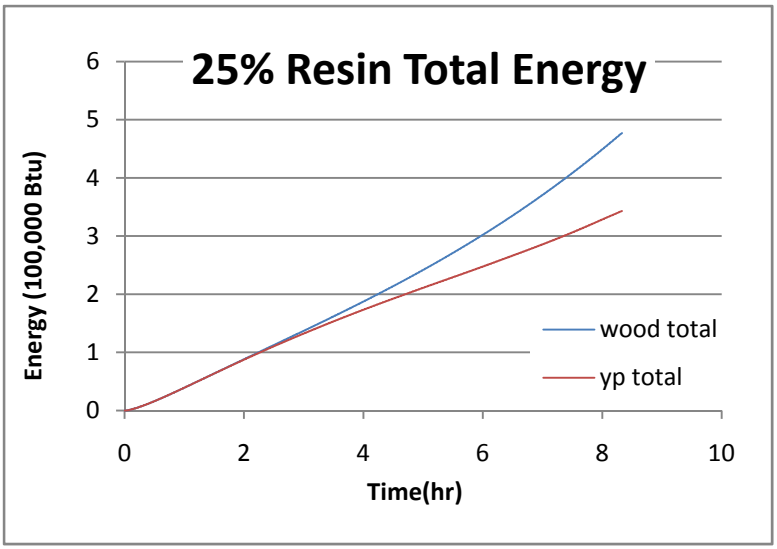


Figure 22 twenty-five percent resin total energy graph

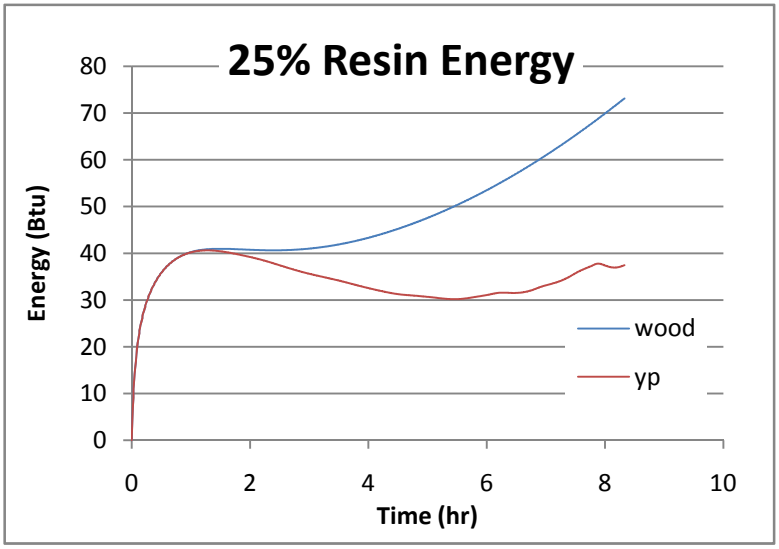


Figure 21 twenty-five percent resin energy difference graph

The yellow pine is below the wood most of the time, most significantly at the peaks. However, it actually takes more energy for the yellow pine when the energy dips.

We also ran the program with 25% of the wood being resin. This graph shows the data for one set of 10,000 iterations. The energy of yellow pine doesn't change until about 2 hours. From then the energy used by yellow pine drops quite a bit and the other wood doesn't. There is about 36 Btus by the end of the iteration. There is quite a difference between this and the *total* resin energy (100,000).

Conclusions

Smart Wall Model

One of the first conclusions we came to was that the melting/freezing point of the yellow pine was actually too high in some cases. It is not possible for the resin to reach the point at which a state change occurs easily or often enough if ever. With a substantial amount of resin (at least 10%) and a 61°F state change point, the yellow pine can make a difference in the heating of a home. There was not a significant discrepancy between the 5% resin yellow pine and regular wood. We also tested 10% and 25% in which there was a difference. The difference was thousands of Btus after about one million iterations. We were not able to run the program enough times to come to many solid conclusions. We do suspect that it would work better in some climates compared to others. It must be able to reach the melting/freezing point often enough to let it store energy, but not so long as to let more heat pass through it.

Model Capabilities

Our program was successful at modeling the heat transfer and the wall with and without yellow pine. A real-time display can give you a good idea of how and where the heat is moving. The studs eventually appear because it is colder where they are and more heat is flowing out of them then the insulation. The resolution is not very fine, but the results are still fairly accurate.

Teamwork

There are only two people on our team, so we didn't have specific roles; the main exception was the programmer. We both helped to do many parts of the project. We thought it was easier to coordinate meeting and working together with two members, rather than a larger group. We had all of the skills we needed, and we didn't want to deal with the conflicting schedules of more people. The research and math we both worked through, usually together, but sometimes on our own. Rachel was the programmer, and Jessie was our artist, and also did a lot of work on the presentations. A great deal of the writing we did at the same time and were able to consult with one another.

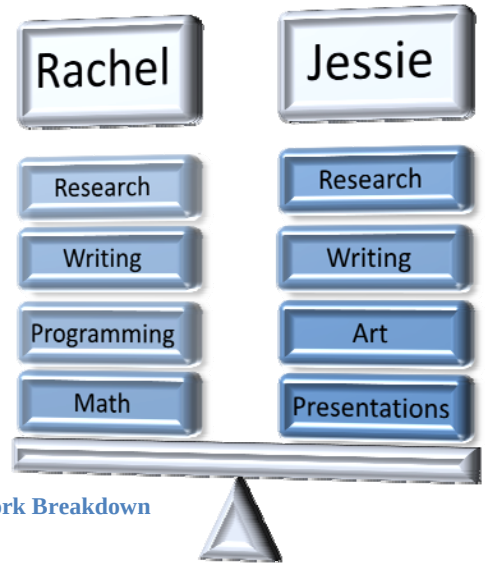


Figure 23 Work Breakdown

Whenever we worked on the project or met with our mentor individually, the one who was present gave the other person a summary of what she had learned. Rachel worked more with the math and Jessie would learn a basic version of it, rather than a detailed account. Jessie would then work on the presentation (poster/keynote) or graphics (report/keynote). We usually met once a week, occasionally more if we had something specific to work on together. While writing the final report we would often work at home and email one another what we had written. After emailing and putting it together, we would review it and send the revised version to the other team mate or go through it with them when we met. When we got together to work on the final report, we would write, put in graphics, review, and have fun.

Recommendations

Model

There are several limitations to our program. There are still some bugs in the program that can make it unstable, usually because of thing like the outside temperature going too high. We also had wanted to work more with the design of the wall. It is difficult to change the design (width or placement of the materials) in the program, so we didn't get to experiment with different sizes and designs of walls. It is not currently interactive, so any changes you make are done in the code itself. We also would have liked to put real temperature and weather histories on the outside and test the wall in an entire year. Since we were only just learning the language and didn't have many expectations, we were able to accomplish most of our goals.

Wall Design

We would like to test the actual designs of the walls, and be able experiment with different widths and arrangements of the materials. We are also interested in passive solar design and putting a design of an entire house together and add in the different components such as windows, roofs, floors, and doors (3-D). There are many different aspects of houses that smart design can be effective in saving energy and keeping the interior comfortable.

Bibliography

1. Ashrae Handbook 1977 Fundamentals. New York, New York: America Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc., 1977.
2. Cooling and Heating: Load Calculation Manual. New York, New York: American Society of Heating, Refrigerating and Air Conditioning Engineers, Inc., 1979.
3. Kernighan, Brian W. and Dennis M. Ritchie. The C Programming Language. Englewood Cliggs, New Jersey: Prentice-Hall, Inc., 1978.
4. Kreith, Frank. Principles of Heat Transfer. 1973. Ed. New York, New York: Intext Educational Publishers, 1973.
5. Mattson, Timothy G., Beverly A. Sanders and Berna L. Massingill. Patterns for Parallel Programming. Boston, Massachusetts: Pearson Education, Inc., 2005.
6. Kuehnel, Paul. "Yellow Pine". August 26, 2007. greenmesh.com/all_posts_from_the_start/home_heatefficiency/
7. "Changes of State". July 22, 2008. id.mind.net/~zona/mstm/physics/mechanics/energy/heatAndTemperature/changesOfPhase/changeOfState.html
8. DoItYourself Inc.."Phase Change Drywall". March 30, 2008. www.doityourself.com/stry/phasechangedrywall
9. "Pinus – The Pines". March 30, 2008 www.ncsu.edu/project/dendrology/index/plantae/vascular/seedplants/gymnosperms/conifers/pine/pinus/pinus.html
10. Ministry of Forests and Range. "Ponderosa or Yellow Pine". March 30, 2008. www.for.gov.bc.ca/hfd/library/documents/treebook/ponderosapine.htm
11. Oliver, William W., Ryker, Russell A.. "Ponderosa Pine". September 10, 1998. www.na.fs.fed.us/SPFO/pubs/silvics_manual/Volume_1/pinus/ponderosa.htm

Acknowledgements

Our Mentor: Bob Robey
Our Family: Peggy Robey, Laura Bohn, Roy Bohn
Our Teacher: LeAnn Salazar
For helping us with the sine wave: Jonathan Robey
For reviewing our proposal: Tom Laub
For reviewing our project: Teri Roberts

Appendix

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <unistd.h>
#include "heat.h"

/**********************************************************************
**********************
 *HEAT -- 2D Heat Transfer Model
 *Rachel Robey, Los Alamos Middle School
 *Copyright, 2007-08

**********************************************************************
**********************/

//define pi
#define Pi 3.14
#define change_energy 2.35

/* Display routines */
void display_init(char *displayname, int iwidth, int iheight);
void display_one_d(int matrix_size_y, int matrix_size_x, double **temp,
int my_offset,
   int mysize, double maxscale, double time, double Temp_max, double
Temp_min);
void set_label(char *text);
void display_close(void);
double state_change_temp(double E, double cv, int j, int i);
double state_change_energy(double T, double cv, int j, int i);
/* Memory allocation routines */
double *dvector(int n);
double **dmatrix(int m, int n);
int **imatrix(int m, int n);

double **Mass, **Energy, **Temperature;   //state variables
int **Material;
double **qkx, **Energyx, **Temperaturex;  //half-step arrays
double **qky, **Energyy, **Temperaturey;  //half-step arrays

int main(int argc, char *argv[]) {
   int rank, size;
   int next, prev;
   int i, j;
   int matrix_size_x, matrix_size_y;
   int ntimes;
   int n;
   int mysize;
   int my_offset;
   double deltaX, deltaY;                  //size of cell
```

```
   double **temp;                          //holder array for display
   double deltat = .0005;                  //hardwired timestep
   double maxScale;
   double time = 0.0;                      //computer simulation time
   double slavetime, totaltime, starttime; //variables to calculate
time taken for the program to run
   double myTE, TotalEnergy, origTE;       //variables for checking
conservation of energy
   double Energy_added = 0.0, Energy_delta = 0.0;
   double Temp_max=65, Temp_min=10;
   FILE *fdata;
   char *desc;                             //variable for labels
   char string[80];
   char *displayname = ":0";
   enum material{OUTSIDE_AIR, INSIDE_AIR, STUCCO, FOAM, WOOD,
INSULATION, SHEETROCK};
   double Material_density[7]= { .076, .076, 116., 2.2, 32., .85, 50.
};
         //density in pounds/ft cubed
   double Material_specific_heat[7]= { .24, .24, .22, .29, .33, .2, .26
};
         //specific heat in Btu/pound*mass*Farenheit
   double Material_conductivity[7]= { 6.0-0.7, 1.24, .5, .2, .8,
.053/5.5,
         1.78/.625 };
         //conductivity in Btu/hour*inch*Farenheit

   MPI_Init(&argc, &argv);
   //Determine size and my rank in MPI_COMM_WORLD communicator
   MPI_Comm_size(MPI_COMM_WORLD, &size) ;
   MPI_Comm_rank(MPI_COMM_WORLD, &rank) ;
   if (argc > 2 && strcmp(argv[1], "-display") == 0) {
      displayname = (char *)malloc(strlen(argv[2]) + 1);
      strcpy(displayname, argv[2]);
   }

   if (display_on)
      display_init(displayname, 416.25, 675);

   if (rank==0)
      printf("Copyright 2008\n");

   //If the process is 0, determine the matrix size and # of iterations
   if (rank == 0) {
       /*
       printf("Matrix Size X : ");
       scanf("%d",&matrix_size_x);
       printf("Matrix Size Y : ");
       scanf("%d",&matrix_size_y);
       printf("Iterations : ") ;
       scanf("%d",&ntimes) ;//
       */
      matrix_size_x = 22;
      matrix_size_y = 34;
      ntimes = 10000000;
   }
```

```
   if (rank==0) {
         fdata=fopen("data.csv","w");
   }
   //Broadcast the size and # of itertations to all processes
   MPI_Bcast(&matrix_size_x, 1, MPI_INT, 0, MPI_COMM_WORLD) ;
   MPI_Bcast(&matrix_size_y, 1, MPI_INT, 0, MPI_COMM_WORLD) ;
   MPI_Bcast(&ntimes, 1, MPI_INT, 0, MPI_COMM_WORLD);

   //Set neighbors
   if (rank == 0)
      prev = MPI_PROC_NULL;
   else
      prev = rank-1;
   if (rank == size - 1)
      next = MPI_PROC_NULL;
   else
      next = rank+1;

   mysize = matrix_size_y/size + ((rank < (matrix_size_y % size)) ? 1 :
0 );
   my_offset = rank * (matrix_size_y/size);
   if (rank > (matrix_size_y % size))
      my_offset += (matrix_size_y % size);
   else
      my_offset += rank;

   if (debug>=1) {
      printf("my rank is %d and mysize is %d\n", rank, mysize);
   }

   /* allocate the memory dynamically for the matrix */

   Mass = dmatrix(mysize+2, matrix_size_x+2);
   Energy = dmatrix(mysize+2, matrix_size_x+2);
   Temperature = dmatrix(mysize+2, matrix_size_x+2);
   Material = imatrix(mysize+2, matrix_size_x+2);

   qkx = dmatrix(mysize+2, matrix_size_x+3);
   Energyx = dmatrix(mysize, matrix_size_x+1);
   Temperaturex = dmatrix(mysize, matrix_size_x+1);

   qky = dmatrix(mysize+3, matrix_size_x+2);
   Energyy = dmatrix(mysize+1, matrix_size_x);
   Temperaturey = dmatrix(mysize+1, matrix_size_x);
   temp = dmatrix(mysize+2, matrix_size_x+2);

   for (j=0; j<=mysize+1; j++) {
      qkx[j][0]=0.0;
      qkx[j][matrix_size_x+2]=0.0;
   }
   for (i=0; i<=matrix_size_x+1; i++) {
      qky[0][i]=0.0;
      qky[mysize+2][i]=0.0;
   }

   if (rank==0&&debug>=1) {
      printf("Memory allocated\n");
```

```
}
//initialize matrix
for (j=0; j<=mysize+1; j++) {
   for (i=0; i<=matrix_size_x+1; i++) {
      Material[j][i]= OUTSIDE_AIR;
   }
}
for (j=0; j<=mysize+1; j++) {
   for (i=10; i<=matrix_size_x+1; i++) {
      Material[j][i]= INSIDE_AIR;
   }
}
for (j=0; j<=mysize+1; j++) {
   Material[j][7]= STUCCO;
}
for (j=0; j<=mysize+1; j++) {
   for (i=8; i<=9; i++) {
      Material[j][i]= FOAM;
   }
}
for (j=0; j<mysize+1; j++) {
      Material[j][10]=WOOD;
}
for (j=0; j<=mysize+1; j++) {
   for (i=11; i<=15; i++) {
      Material[j][i]= INSULATION;
   }
}
for (j=1; j<=2; j++) {
   for (i=11; i<=15; i++) {
      Material[j][i]= WOOD;
   }
}
for (j=17; j<=18; j++) {
   for (i=11; i<=15; i++) {
      Material[j][i]= WOOD;
   }
}
for (j=33; j<=34; j++) {
   for (i=11; i<=15; i++) {
      Material[j][i]= WOOD;
   }
}
for (j=0; j<=mysize+1; j++) {
   Material[j][16]= SHEETROCK;
}
for (j=0; j<=mysize+1; j++) {
   for (i=0; i<=6; i++) {
      Temperature[j][i]= Temp_max;
   }
}
for (j=0; j<=mysize+1; j++) {
   for (i=7; i<=16; i++) {
      Temperature[j][i]= 63.;
   }
}
for (j=0; j<=mysize+1; j++) {
```

```
      for (i=16; i<=23; i++) {
         Temperature[j][i]= 70.;
      }
   }

   for (j=0; j<=mysize+1; j++) {
      for (i=0; i<=matrix_size_x+1; i++) {
         Mass[j][i]= (Material_density[ Material[j][i] ])/1728.;
            //multiply by 1 foot/12 inches three times to cancel out
units, becomes pounds,mass
         if (yellow_pine){
           if(Material[j][i]==WOOD){
             Energy[j][i]=state_change_energy(Temperature[j][i],
Material_specific_heat[Material[j][i]], j, i);
           }
           else {
               Energy[j][i]= (Material_specific_heat[ Material[j][i]
])*Temperature[j][i];
                          //specific energy is in Btu/pound,mass
           }
         }
         else {
             Energy[j][i]= (Material_specific_heat[ Material[j][i]
])*Temperature[j][i];
         }

      }
   }

   deltaX=1.0;
   deltaY=1.0;
   if (rank==0&&debug>=1) {
      printf("initial values set\n");
   }
   //display initial values
   if (display_on) {
      maxScale=.07;
      desc="wall materials by density";
      set_label(desc);
      display_one_d(matrix_size_x, matrix_size_y, Mass, my_offset,
mysize,
            maxScale, time, Temp_max, Temp_min);
   }
   sleep(10);

   myTE=0.0;
   for (j=1; j<=mysize; j++) {
      for (i=1; i<=matrix_size_x; i++) {
         myTE+=Energy[j][i];
      }
   }
   MPI_Allreduce(&myTE, &origTE, 1, MPI_DOUBLE, MPI_SUM,
MPI_COMM_WORLD);

   if (rank==0&&debug>=1) {
      printf("initial values displayed\n");
   }
```

```
   //run the simulation for given number of iterations/
   starttime = MPI_Wtime() ;
   for (n = 0; n < ntimes; n++) {
      MPI_Request req[8];
      MPI_Status status[8];

      //Send and receive boundary information
      MPI_Isend(Energy[1], matrix_size_x+2, MPI_DOUBLE, prev, 1,
            MPI_COMM_WORLD, req);
      MPI_Irecv(Energy[mysize+1], matrix_size_x+2, MPI_DOUBLE, next, 1,
            MPI_COMM_WORLD, req+1);

      MPI_Isend(Energy[mysize], matrix_size_x+2, MPI_DOUBLE, next, 2,
            MPI_COMM_WORLD, req+2);
      MPI_Irecv(Energy[0], matrix_size_x+2, MPI_DOUBLE, prev, 2,
            MPI_COMM_WORLD, req+3);

      if (rank==0&&debug>=1) {
         printf("values for energy communicated\n");
      }

      MPI_Isend(Temperature[1], matrix_size_x+2, MPI_DOUBLE, prev, 5,
            MPI_COMM_WORLD, req+4);
      MPI_Irecv(Temperature[mysize+1], matrix_size_x+2, MPI_DOUBLE,
next, 5,
            MPI_COMM_WORLD, req+5);

      MPI_Isend(Temperature[mysize], matrix_size_x+2, MPI_DOUBLE, next,
6,
            MPI_COMM_WORLD, req+6);
      MPI_Irecv(Temperature[0], matrix_size_x+2, MPI_DOUBLE, prev, 6,
            MPI_COMM_WORLD, req+7);

      if (rank==0&&debug>=1) {
         printf("values for temperature communicated\n");
      }
      MPI_Waitall(8, req, status);
      if (rank==0&&debug>=1)
         printf("Communication successful\n");
      //set boundary conditons
      Energy_delta=0.0;
      for (j=0; j<=mysize+1; j++) {
         Energy[j][0]=2.4;
         Temperature[j][0]=(((Temp_max+Temp_min)/2))+((Temp_max-
Temp_min)/2)*(sin((time+360)*Pi/720));
         Temperature[j][matrix_size_x+1]=70.;

Energy_added+=Material_specific_heat[Material[j][matrix_size_x+1]]*70.0
-Energy[j][matrix_size_x+1];

Energy_delta+=Material_specific_heat[Material[j][matrix_size_x+1]]*70.0
-Energy[j][matrix_size_x+1];
         Energy[j][matrix_size_x+1]= (Material_specific_heat[
Material[j][matrix_size_x+1] ])*70.0;
      }

      for (i=0; i<=matrix_size_x+1; i++) {
```

```
         if (my_offset==0) {
            Energy[0][i]=Energy[1][i];
            Temperature[0][i]=Temperature[1][i];
         }
      }
      if (matrix_size_y==my_offset+mysize) {
         Energy[mysize+1][i]=Energy[mysize][i];
         Temperature[mysize+1][i]=Temperature[mysize][i];
      }
      if (rank==0&&debug>=1)
         printf("Boundary conditions set\n");

      //set timestep
      // We should calculate a timestep here
      if (rank==0&&debug>=1) {
         printf("deltat set to %20.1f\n", deltat);
      }

      time+=deltat;
      //For each element of the matrix ...
      if (rank == 0&&debug>=1) {
         printf("Before 1st pass\n");
      }
      //first pass
      //x direction
      for (j = 0; j <= mysize+1; j++) {
         for (i = 1; i<=matrix_size_x+1; i++) {

            qkx[j][i]=-Material_conductivity[ Material[j][i-1]
]*(Temperature[j][i-1]-Temperature[j][i])/.5/144./60.
               *deltat+ -Material_conductivity[ Material[j][i ]
]*(Temperature[j][i-1]-Temperature[j][i])/.5/144./60.*deltat;
            // multiply negative conductivity of material be the
differnce in temperatures, divide by .5 for the distance
            //of heat flow, division by 144 and 60 cancels units
multiply by the time step

            if (Material[j][i]!=OUTSIDE_AIR&&Material[j][i-
1]==OUTSIDE_AIR) {
               //calculate convection and radiation between air and
surface, conductance is in Btu/hr/ft squared/F
               qkx[j][i]-=6.0*deltat*(Temperature[j][i-1]-
Temperature[j][i])/60./144.;
            }
            if (Material[j][i-
1]!=INSIDE_AIR&&Material[j][i]==INSIDE_AIR) {
               qkx[j][i]-=1.46*deltat*(Temperature[j][i-1]-
Temperature[j][i])/60./144.;
            }
         }
      }

      if (rank==0&&debug>=1) {
         printf("First pass x direction complete\n");
      }

      //y direction
```

```
      for (j = 1; j<=mysize; j++) {
         for (i=0; i<=matrix_size_x+1; i++) {
            qky[j][i]=-Material_conductivity[ Material[j-1][i]
]*(Temperature[j-1][i]-Temperature[j][i])/.5/144./60.
               *deltat+ -Material_conductivity[ Material[j ][i]
]*(Temperature[j-1][i]-Temperature[j][i])/.5/144./60.*deltat;
            //multiply negetive conductivity of material be the
differnce in temperatures, divide by .5 for the distance
            //of heat flow, division by 144 and 60 cancels units
multiply by the time step
         }
      }
      if (rank==0&&debug>=1) {
         printf("First pass complete\n");
      }

      //second pass
      if (rank==0&&debug>=1) {
         printf("Second Pass started\n");
      }
      for (j = 0; j <=mysize+1; j++) {
         for (i = 0; i<=matrix_size_x+1; i++) {
            Energy[j][i] = Energy[j][i]+(qkx[j][i+1]-
qkx[j][i]+qky[j+1][i]-qky[j][i])/Mass[j][i];
            if (yellow_pine){
               if(Material[j][i]==WOOD){
                  Temperature[j][i]=state_change_temp(Energy[j][i],
Material_specific_heat[Material[j][i]], j, i);
               }
               else {
                  Temperature[j][i]=
Energy[j][i]/(Material_specific_heat[Material[j][i]]);
               }
            }
            else {
                  Temperature[j][i]=
Energy[j][i]/(Material_specific_heat[Material[j][i]]);
            }
         }
      }
      if (rank==0&&debug>=1) {
         printf("Second pass complete\n");
      }

      if (rank==0&&debug>=1) {
         printf("Done calculations\n");
      }
      //set temp to ____ and display
      if (debug >= 2) {
         for (j=0; j<=mysize+1; j++) {
            for (i=0; i<=matrix_size_x+1; i++) {
               printf("end of cycle %d %d %lf %lf %lf\n", i, j,
                     Mass[j][i], Energy[j][i], Temperature[j][i]);
            }
         }
      }
      if (display_on) {
```

```
         maxScale=72.0;
         desc="Temperature";
         sprintf(string, "%s     iter %d      time %.2lf", desc, n,
time);
         set_label(string);
         if (n%display_on == 0) {
            display_one_d(matrix_size_x, matrix_size_y, Temperature,
                  my_offset, mysize, maxScale, time, Temp_max,
Temp_min);
         }
      }

      myTE=0.0;
      TotalEnergy=0.0;
      for (j=1; j<=mysize; j++) {
         for (i=1; i<=matrix_size_x; i++) {
            if (isnan(Energy[j][i])) {
               printf("Error -- Energy[%d][%d]=%f\n", i, j,
Energy[j][i]);
            }
            myTE+=Energy[j][i];
         }
      }

      MPI_Allreduce(&myTE, &TotalEnergy, 1, MPI_DOUBLE, MPI_SUM,
            MPI_COMM_WORLD);
      /*
      if(((fabs(TotalEnergy-origTE)>1.0E-
6)||isnan(TotalEnergy))&&check==1){
      printf("Conservation of energy\nEnergy difference:%e\n",
TotalEnergy-origTE);
      printf("Problem occured on iteration %5.5d at time %f.\n", n,
time);
      exit(0);
      }
      */

      if (symmetry_check==1) {
         for (j=1; j<=mysize/2; j++) {
            for (i=1; i<=matrix_size_x; i++) {
               if (Temperature[j][i]!=Temperature[mysize+1-j][i]) {
                  printf("Temp %d %d=%lf   Temp%d %d=%lf\n", j, i,
                        Temperature[j][i], mysize+1-j, i,
                        Temperature[mysize+1-j][i]);
                  printf("Energy %d %d=%lf   Energy %d %d=%lf\n", j, i,
                        Energy[j][i], mysize+1-j, i,
                        Energy[mysize+1-j][i]);
               }
            }
         }
      }

      //print iteration info
      if (display_on && n%display_on == 0) {
         if (rank == 0) {
            printf(
```

```
                  "Iteration:%5.5d, Time:%f, Timestep:%f Total
energy:%f  Energy added: %f  Energy delta: %f\n",
                  n, time, deltat, TotalEnergy, Energy_added,
Energy_delta);
            fprintf(fdata, "%5.5d, %13.5f,  %13.5f,  %13.8f\n",
                              n, Energy_added, time/60.0, Energy_delta);

         }

      }

   } // End of iteration loop

   //Compute the average time taken/processor/
   slavetime = MPI_Wtime() - starttime;
   MPI_Reduce(&slavetime, &totaltime, 1, MPI_DOUBLE, MPI_SUM, 0,
         MPI_COMM_WORLD);

   //Print the total time taken
   if (rank == 0)
      printf("Energy added   %lf Btus  Time  %lf minutes\n",
Energy_added
            *Mass[j][matrix_size_x+1], time);
   printf("[%d] Flow finished in %lf seconds\n", rank,
totaltime/(double)size);//*/

   if (display_on)
      display_close();

   fclose(fdata);
   //Should free memory allocated with dmatrix call
   MPI_Finalize();
   exit(0);

}

double state_change_energy(double T, double cv, int j, int i) {
      double energy_return;
      if(T<61.){
            energy_return=cv*T;
      }
      else if(T>62.){
                  energy_return=cv*(T-1.)+(change_energy);
            }
      else {
            energy_return=cv*61.+((T-61.)*change_energy);
      }

      return(energy_return);
}
double state_change_temp(double E, double cv, int j, int i) {
      double temp_return;
      if(E<(cv*61.)){
            temp_return=Energy[j][i]/cv;
      }
      else if(E>(cv*61.+change_energy)){
            temp_return=(E-change_energy)/cv+1.;
```

```
      }
      else {
            temp_return=((E-cv*61.)*change_energy)+61.;
      }
      return(temp_return);
}
```

Graphics

```
#include <mpi.h>
#define MPE_GRAPHICS
#include "mpe.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include "heat.h"

#define Pi 3.14


int width=750;
int height=750;
MPE_XGraph graph;
MPE_Color *color_array;
int ncolors=256;
char *label;
void display_init(char *displayname, int iwidth, int iheight){

  int ierr;
  int rank;

  //int ncolors2;

 /* Open the graphics display */

 width=iwidth;
 height=iheight;

  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPE_Open_graphics( &graph, MPI_COMM_WORLD, displayname,
               -1, -1, width, height+10, 0 );

  color_array = (MPE_Color *) malloc(sizeof(MPE_Color)*ncolors);

  ierr = MPE_Make_color_array(graph, ncolors, color_array);
  if (ierr && rank == 0) printf("Error(Make_color_array): ierr is
%d\n",ierr);

  //MPE_Num_colors(graph, &ncolors2);
  //printf("size of color array is %d\n",ncolors2);

}
```

```
 void display_close(void){
   MPE_Close_graphics(&graph);
 }

void set_label(char *text)
{
      label=text;
}
void display_one_d(int matrix_size_x, int matrix_size_y, double **temp,
   int my_offset, int mysize, double maxscale, double time, double
Temp_max, double Temp_min)
{
  int i, j;
  unsigned int plot_value;
  char string[60], outside_temp[12];
  int daytime, hourtime, minutetime, am_pm;
  /*double localmax=0;
  double localmin=2000;
  for(j=0;j<=mysize;j++){
    for(i=0;i<=matrix_size_x;i++){
      if(temp[j][i]>localmax)
        localmax=temp[j][i];
      if(temp[j][i]<localmin)
        localmin=temp[j][i];
    }
  }
  printf("localmax %f localmin %f\n", localmax, localmin);//*/
  for(j=1;j<=mysize;j++){
      for(i=1;i<=matrix_size_x;i++){
       int xloc, yloc, xwid, ywid;
        xloc = ((i - 1) * width) / matrix_size_x;
        yloc = ((my_offset + j - 1) * height/2) / matrix_size_y;
        xwid = (i * width) / matrix_size_x - xloc;
        ywid = ((my_offset + j) * height/2) / matrix_size_y - yloc;
        plot_value = ncolors - ((double)ncolors*temp[j][i]/maxscale) +
2;
        //printf("temp[%d][%d]=%lf\n",i,j,temp[j][i]);
        if (plot_value < 2) plot_value = 2;
        if (plot_value > ncolors) plot_value = ncolors;
        //printf("%d %d %d %8.5f\n",i,j,plot_value,temp[i][j]);
        if(isnan(temp[j][i]))MPE_Fill_rectangle( graph, xloc, yloc,
xwid, ywid, MPE_WHITE);
        else if(temp[j][i]<0)MPE_Fill_rectangle( graph, xloc, yloc,
xwid, ywid, MPE_WHITE);
        else if(temp[j][i]>maxscale)MPE_Fill_rectangle( graph, xloc,
yloc, xwid, ywid, MPE_BLACK);
        else MPE_Fill_rectangle(graph, xloc, yloc, xwid, ywid,
color_array[plot_value]);
   }
  }//*/
  //if(debug==1){printf("Color display complete\n");}
  MPE_Fill_rectangle(graph, 0, height/2, width, height, MPE_WHITE);
  MPE_Fill_rectangle(graph, 0, height/2, width, 2, MPE_BLACK);
  for(i=1;i<=matrix_size_x;i++){
      int xloc, xwid;
        xloc = ((i - 1) * width) / matrix_size_x;
        xwid = (i * width) / matrix_size_x - xloc;
```

```
        MPE_Fill_rectangle( graph, xloc, ((height)-
(((height/2.1))*(temp[1][i])/(maxscale))), xwid, 1, color_array[1]);
  }//*/
  if(strncmp(label, "wall", 4)){
  MPE_Fill_rectangle(graph, .285*width, 0, 1., height, MPE_BLACK);
  MPE_Fill_rectangle(graph, .73*width, 0, 1., height, MPE_BLACK);
  MPE_Fill_rectangle(graph, 0, .5375*height, 15., 1., MPE_BLACK);
  MPE_Draw_string(graph, .01*width, .5375*height, MPE_BLACK, "70");
  MPE_Fill_rectangle(graph, 0, .6037*height, 15., 1., MPE_BLACK);
  MPE_Draw_string(graph, .01*width, .6037*height, MPE_BLACK, "60");
  MPE_Fill_rectangle(graph, 0, .669*height, 15., 1., MPE_BLACK);
  MPE_Draw_string(graph, .01*width, .669*height, MPE_BLACK, "50");
  MPE_Fill_rectangle(graph, 0, .735*height, 15., 1., MPE_BLACK);
  MPE_Draw_string(graph, .01*width, .735*height, MPE_BLACK, "40");
  MPE_Fill_rectangle(graph, 0, .8025*height, 15., 1., MPE_BLACK);
  MPE_Draw_string(graph, .01*width, .8*height, MPE_BLACK, "30");
  MPE_Fill_rectangle(graph, 0, .8675*height, 15., 1., MPE_BLACK);
  MPE_Draw_string(graph, .01*width, .8675*height, MPE_BLACK, "20");
  MPE_Fill_rectangle(graph, 0, .9345*height, 15., 1., MPE_BLACK);
  MPE_Draw_string(graph, .01*width, .9345*height, MPE_BLACK, "10");
  if( ( ( (int)time+360 )/60/12)%2) {
        MPE_Fill_circle(graph, .85*width, .93*height, 35, MPE_BLACK);
        MPE_Fill_circle(graph, .88*width, .93*height, 30, MPE_WHITE);
  }
  else{
        MPE_Fill_circle(graph, .85*width, .93*height, 35, 60);
  }

  daytime=(int)time%1440;
  hourtime=((int)(time+660.0)/60)%12+1;
  am_pm=hourtime/12%2;
  minutetime=(int)time%60;
  if(am_pm==0){
  sprintf(string,"%d:%02d pm",hourtime,minutetime);
  }
  else{
  sprintf(string,"%d:%02d am",hourtime,minutetime);
  }
  MPE_Draw_string(graph, .835*width, .92*height, MPE_BLACK, string);
  sprintf(outside_temp, "%.2lf", (((Temp_max+Temp_min)/2))+((Temp_max-
Temp_min)/2)*(sin((time+360)*Pi/720)));
  MPE_Draw_string(graph, .835*width, .94*height, MPE_BLACK,
outside_temp);

}

  //if(debug==1){printf("Graph display complete\n");}
  MPE_Fill_rectangle(graph, 0, height, width, height, MPE_WHITE);
  if(my_offset==0)MPE_Draw_string( graph, 0, height+10, MPE_BLACK,
label);
  MPE_Update( graph );
  sleep(wait_time);
}
```