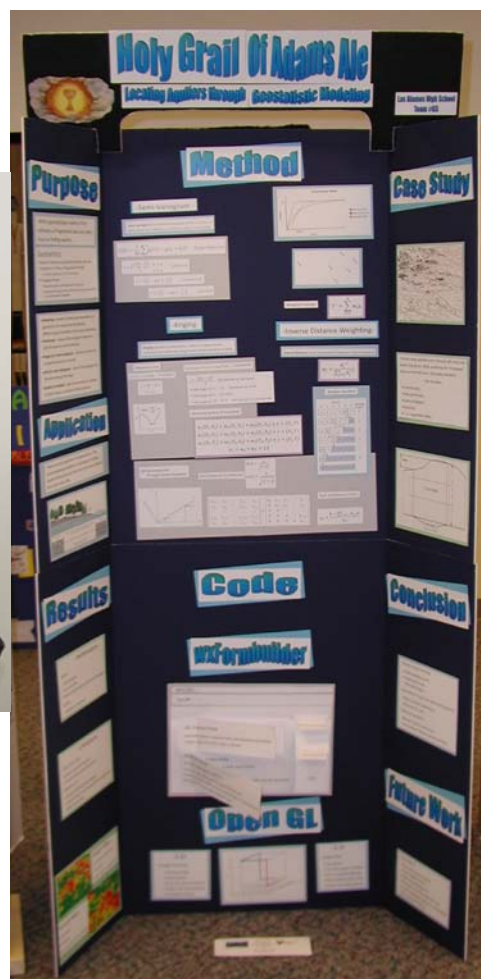


Supercomputing Challenge

Finalist Reports

2009-2010



www.challenge.nm.org

Printed in cooperation with
Los Alamos National Laboratory
High Performance Computing Group (HPC-3)
and
New Mexico Public Education Department

Cover: **The Holy Grail of Adam's Ale**
Team 65 from Los Alamos High School
Rachel Robey, Gabriel Montoya,
Orion Staples and Orli Shlachter
Teacher Lee Goodwin
Winner in the Technical Poster Competition



Notification: These final reports are presented in an abridged form, leaving out actual code, color, and appendices where appropriate. Complete copies of most of the final reports are available from the archives of Supercomputing Challenge web site:
<http://www.challenge.nm.org> .

New Mexico Supercomputing Challenge **2009 – 2010 Finalist Reports**

Table of Contents

About the New Mexico Supercomputing Challenge

For more information, please visit our website at <http://www.challenge.nm.org>2

2009—2010 Challenge Awards4

Scholarship winners8

Sponsors9

Participants10

Judges.....16

Finalist Reports19

1. Control and Spread of Wildfires II, Melrose High, Team 99
2. The Holy Grail of Adam’s Ale, Locating Aquifers through Geostatistic Modeling, Los Alamos High, Team 65
3. To Kill a Flocking Bird, Los Alamos High, Team 70
4. The Metropolis Algorithm and Nanometer-Scale pattern Formation, Albuquerque Academy, Team 5
5. Arbitrary Precision Integers on the Cell Processor, Desert Academy, Team 36
6. The Spread of the Black Death in London, Desert Academy, Team 37
7. Enlightenment: Metropolis-Hastings Ray Prediction Model in 3D Space, Los Alamos High, Team 69
8. Particle-Atmosphere Interaction, McCurdy High, Team 84



Supercomputing Challenge Vision

The Vision of the Supercomputing Challenge is to be a nationally recognized program that promotes computational thinking in science and engineering so that the next generation of high school graduates is better prepared to compete in an information based economy.

Supercomputing Challenge Mission

The Mission of the Supercomputing Challenge is to teach teams of middle and high schools students how to use powerful computers to analyze, model and solve real world problems.

About the Supercomputing Challenge

The Supercomputing Challenge (the Challenge) is an exciting program that offers a truly unique experience to students in our state. The opportunity to work on the most powerful computers in the world is currently available to only a very few students in the entire United States, but in New Mexico, it is just one of the benefits of living in the "Land of Enchantment."

The Challenge is a program encompassing the school year in which teams of students complete science projects using high-performance supercomputers. Each team of up to five students and a sponsoring teacher defines and works on a single computational project of its own choosing. Throughout the program, help and support are given to the teams by their project advisors and the Challenge organizers and sponsors.

The Challenge is open to all interested students in grades 6 through 12 on a nonselective basis. The program has no grade point, class enrollment or computer experience prerequisites. Participants come from public, private, parochial and home-based schools in all areas of New Mexico. The important requirement for participating is a real desire to learn about science and computing.

Challenge teams tackle a range of interesting problems to solve. The most successful projects address a topic that holds great interest for the team. In recent years, ideas for projects have come from Astronomy, Geology, Physics, Ecology, Mathematics, Economics, Sociology, and Computer Science. It is very important that the problem a team chooses is what we call "real world" and not imaginary. A "real world" problem has measurable components. We use the term Computational Science to refer to science problems that we wish to solve and explain using computer models.

Those teams who make significant progress on their projects can enter them in the competition for awards of cash and scholarships for the individuals and computer equipment for the school. Team trophies are also awarded for: Teamwork, Best Written Report, Best Professional Presentation, Electronic Search & Browse, Creativity and Innovation, Environmental Modeling, High Performance, Science is Fun and the Judges' Special Award, just to name a few.

The Challenge is offered at minimal cost to the participants or the school district. It is sponsored by a partnership of federal laboratories, universities, and businesses. They provide food and lodging for events such as the kickoff conference during which students and teachers are shown how to use supercomputers, learn programming languages, how to analyze data, write reports and much more.

These sponsors also supply time on the supercomputers and lend equipment to schools that need it. Employees of the sponsoring groups conduct training sessions at workshops and advise teams throughout the year. The Challenge culminates with an Expo and Awards Ceremony in the spring at Los Alamos National Laboratory.

History

The New Mexico High School Supercomputing Challenge was conceived in 1990 by former Los Alamos Director Sig Hecker and Tom Thornhill, president of New Mexico Technet Inc., a nonprofit company that in 1985 set up a computer network to link the state's national laboratories, universities, state government and some private companies. Sen. Pete Domenici, and John Rollwagen, then chairman and chief executive officer of Cray Research Inc., added their support.

In 2001, the Adventures in Supercomputing program formerly housed at Sandia National Laboratories and then at the Albuquerque High Performance Computing Center at the University of New Mexico merged with the former New Mexico High School Supercomputing Challenge to become the New Mexico High School Adventures in Supercomputing Challenge.

In 2002, the words "High School" were dropped from the name as middle school teams had been invited to participate in 2000 and had done well.

In the summer of 2005, the name was simplified to the Supercomputing Challenge.

In 2007, the Challenge began collaborating with the middle school Project GUTS, (Growing Up Thinking Scientifically), an NSF grant housed at the Santa Fe Institute.

2009—2010 Challenge Awards



Melrose High trio named top team in 20th New Mexico Supercomputing Challenge

Student research project modeled behavior of wildfire

LOS ALAMOS, New Mexico, April 27, 2010—A trio of students from Melrose High School captured the top prize in the 20th New Mexico Supercomputing Challenge hosted by Los Alamos National Laboratory. The report "Control and Spread of Wildfires II" by

brothers Richard and Randall Rush and Kyle Jacobs built upon previous research by the team and added a new variable, topography, to the computer model as a factor contributing to the behavior of wildfire.

Each student receives a check for \$1,000. The team also received the Crowd Favorite Award—and \$100—as selected by student participants, teachers, and mentors.

Two Los Alamos High School teams captured second and third place. "The Holy Grail of Adam's Ale" received second place, and "To Kill a Flocking Bird" captured the third-place prize.

The [Supercomputing Challenge](#) is open to any New Mexico high-school or middle-school student. More than 250 students representing 70 teams from schools around the state spent the school year researching scientific problems, developing sophisticated computer programs, and learning about computer science with mentors from the state's national laboratories and other organizations.

The goal of the yearlong event is to teach teams of middle- and high-school students how to use powerful computers to analyze, model, and solve real-world problems. Participating students improve their understanding of technology by developing skills in scientific inquiry, modeling, computing, communications, and teamwork.

The Los Alamos High School team of Gabriel Montoya, Rachel Robey, Orli Shlachter, and Orion Staples each received \$500 for the second-place research project, which used geostatistics, a branch of applied statistics, to find aquifers and other groundwater sources. Robey and Montoya took third place in last year's challenge for their project on energy efficiency through smart wall design.

The team also received the Best Technical Poster Award. Their poster will be used on the front cover for the 2009-10 final reports book, which will be published this fall during the kickoff for the 2010-11 Supercomputing Challenge. The team also received the Visualization Award from New Mexico Institute of Mining and Technology. The award comes with \$150.

The third-place team consists of students Peter Ahrens, Stephanie Djidjev, Vickie Wang, and Mei Lui. Their project explored techniques used to optimize the parameters of flocking, a phenomenon frequently exhibited by birds during migration, animals such as elephants who flock to protect smaller, weaker members, and in humans. They each receive \$250.

The quartet of Los Alamos High students also received the Best Internet Research Prize—and a \$500 cash award—from the Council for Higher Education Computing/Communication Services. They also garnered the New Mexico Network for Women in Science and Engineering award for best project with a majority of women team members, and shared the Visualization Award with the second-place team from Los Alamos High. The award comes with \$150.



Additional Finalist teams were:

Team 5, Albuquerque Academy, The Metropolis Algorithm and Nanometer-Scale Pattern Formation

Team Members: Michael Wang, Jack Ingalls

Sponsor: Jim Mims, Mentor: David Dunlap, Ph.D

Team 36, Desert Academy, Arbitrary Precision Integers on the Cell Processor

Team Members: Megan Belzner, Matt Rohr, Bjorn Swenson

Sponsor/Mentor: Thomas Christie

Team 37, Desert Academy, The Spread of the Black Death in London

Team Members: Katie Boot, Sara Hartse

Sponsors: Thomas Christie, Jocelyn Comstock

Team 69, Los Alamos High School,

Enlightenment: Metropolis-Hastings Ray Prediction Model in 3D Space

Team Members: Kathy Lin, Jake Poston, Ryan Marcus, Dov Shlachter

Mentor: Lee Goodwin

Team 84, McCurdy High School, Particle-Atmosphere Interaction

Team Members: Dennis Trujillo, Oliver Galvan, Brandon Ricci

Mentors: John Pretz, Brenda Dingus, Philip Sanchez

All were taking home posters for their school trophy cabinets and a large banner for their gym and \$50 per student.

The Challenge honored Erika DeBenedictis who won the individual Intel Science Talent Search with \$100,000 scholarship. Erika participated as the youngest finalist judge as part of the Challenge's appreciation for her skills.

The Creativity and Innovation Award, \$100, from Sandia National Laboratories went to Team 33, Deming High School, Tortuga Trouble: A New Survey Method. Team Members were Rocky Navarrete and Gabriela Anguiano. Their teacher sponsor was Creighton Edington.

The Science Rocks award went home with Team 18, Aspen Elementary, The Disappearing Honeybees. Team Members were Kim Vo, Rowan Cantua and Kaelan Prime. Their sponsor was Zeynep Unal. Their mentor was Duc Vo.

The best epidemiology project was given to Team 37 from Desert Academy. The title of their project was The Spread of the Black Death in London. Team members were Katie Boot and Sara Hartse. Their sponsors were Thomas Christie and Jocelyn Comstock.

Team 79, Los Alamos Middle School, Alien River Cloggers, won the Los Alamos National Laboratory Environmental Modeling Award and \$100. Team members were Jacob Holesinger and Kevin Tao, Teacher sponsor was Clara Vigil. Their Mentor was Terry Holesinger.

Cray, Inc. awarded Team 36, Desert Academy, Arbitrary Precision Integers on the Cell Processor the High Performance Computing Award. Team members were Megan Belzner, Matt Rohr, Bjorn Swenson. Sponsor and mentor was Thomas Christie.

The Don't Panic Award goes Team 50, Freedom High School, A Proper Interpretation of Panic. Team members were William Barrett, Holly Campbell, Chelsea Kibbee and Amy Ronan. Their teacher sponsor was Joe Vertrees.

Teamwork Award went to 107 from Quemado High and Silver High Their project First Impressions earned \$100. Team members were Jose Mora and Austin Nightengale. Their teachers were Laura Larisch and Peggy Larisch.

Team 13 from Artesia received \$100 for the Best Web Presentation of a Final Report for their project Classroom Behavior. Team members were Cristina Villa, Nayeli Ramirez and Brenna Arredondo. Teacher sponsor was Amy Mathis. Mentors were Randall Gaylor, Nick Bennett and Olivia Rueda.

The Award for Best Agent Based Modeling goes to team 37 from Desert Academy. The team members, Katie Boot and Sara Hartse, worked on the project, The Spread of the Black Death in London. The sponsors were Thomas Christie and Jocelyn Comstock.

Team 102, from Navajo Preparatory School, Alexis Archambault, Ariel Nephew and Wilfred Jumbo received the Community Focus Award and \$100 for their project The Latest Buzz About Bees. Their sponsor was Mavis Yazzie.

The Award for Best Professional Presentation Award, given by the Albuquerque Journal, went to Team 68 from Los Alamos High School. They were well-prepared, articulate, dressed appropriately, and responsive to feedback. The team members were Sam Baty and Peter Armijo. Their sponsors were Lee Goodwin and Diane Medford. Mentors were Roy Baty and John Armijo. Their project title was Astrophysical N-Body Simulations of Star Clusters.

Team 53 was taking home the Jeff Bingaman Middle School Award. Team Members were Ariel Koh and Aline Parlman Their project was Grocery Tracker. Their mentor was Dr. Aik-Siong Koh.

Harry Henderson and JP Gonzales were awarded the Challenge Service Award for traveling the state to support teams and being a vital part of the Challenge management team.

Two new awards went to Jerry Esquivel, CEPi1, an Albuquerque charter school and Laura Larisch, Quemado High. Laura won the Newcomer Award for being a first year teacher sponsor who supported three teams and two scholarship applicants. Jerry won the Magellan Award for bringing computer science to his school community and modeling lifelong learning.

The best logo award went to Team 11, for the best graphic for next year's logo which goes on the t-shirts and teacher bags. Artesia High School, Sugarscaping On a Beowulf Ring. Team Members were Wesley Green, Isaiah Jordan, James McGee, Wen Hai Zheng. Their teacher sponsor was Randall Gaylor. Their mentors were Nick Bennett and Jose Quiroz.

The Best Use of Parallelism Award went to Team 69, Los Alamos High School, with their project Enlightenment: Metropolis-Hastings Ray Prediction Model in 3D Space. Team members were Kathy Lin, Jake Poston, Ryan Marcus and Dov Shlachter. Their Teacher sponsor was Lee Goodwin. They received a \$300 cash award from the Computer Science and Engineering Department at New Mexico Tech.

A total of \$62,700 in individual scholarships—\$50,000 from the Laboratory's Computer, Computational, and Statistical Sciences Division—were awarded on Tuesday at Los Alamos. An additional \$2500 came from Intel, \$1200 from the Challenge for the Willard Smith Scholarships and \$9,000 was given by in-state colleges and universities. Students receiving scholarships were: Erika DeBenedictis, Albuquerque Academy, Dennis Trujillo, McCurdy High, Gabriela Anguiano, Deming High, Brenna Arredondo, Artesia, Kathy Lin, Los Alamos, Oliver Galvan, McCurdy, Jon



Romero, Bloomfield High, Cristina Villa, Artesia High, Jack Ingalls, Albuquerque Academy, Ryan Marcus, Los Alamos High, Rocky Navarrete, Deming High, Michael Wang, Albuquerque Academy, Elysia Berg, Hope Christian High, William Jennings, Hope Christian High, Judith Flores, Quemado High, Janessa Larabee, Quemado, Erick Chavez, Deming High, Cameron Corley, Bloomfield High, Matt Crockett, Bloomfield High, Kaitlyn Dow, Northern New Mexico College, Amanda Edington, Deming High, Yolly Gamboa, Hatch High, Jake Poston, Los Alamos High, Brandon Ricci, McCurdy High, and Jeremy Salazar, Bloomfield High.

Consult, the Challenge management team, honored head finalist judge, Michael Trahan, Sandia National Labs, veteran teacher, Karen Glennon, Jackson Mid School, Albuquerque, and new teacher sponsor, Creighton Eddington, Deming High, for their dedication, time and expertise.

CHECS, the New Mexico Council for Higher Education Computing/Communication Service, provided cash for random drawing door prizes and Amy Ronan from Freedom High, Michael Wang from Albuquerque Academy, Kelsey Theriot from Jackson Middle School, Ryan Cortez and Lewis Taylor from V. Sue Cleveland High and Ryan Marcus from Los Alamos High each received \$100.

Students presented their research to a team of volunteer judges on Monday at the Lab's J. Robert Oppenheimer Study Center and discussed poster displays of their computing projects. They also toured the Laboratory's supercomputing centers and heard talks and saw demonstrations by Laboratory researchers.

Sponsors

The Supercomputing Challenge is sponsored by Los Alamos and Sandia national laboratories and the state of New Mexico.

Educational partners include The Center for Connected Learning, CHECS, Eastern New Mexico University, High Plains Regional Cooperative, MIT StarLogo, New Mexico Computing Applications Center, New Mexico Highlands University, New Mexico Institute of Mining and Technology, Northern New Mexico College, New Mexico Public Education Department, New Mexico State University, San Juan College, Santa Fe Community College, Santa Fe Institute, and the University of New Mexico.

Lockheed Martin, Los Alamos National Laboratory Foundation, The Math Works, Synergy Group, Vandyke Software Inc., and Wolfram Research, Inc. are "Gold" commercial partners. "Silver" commercial partners are Abba Technologies, Google RISE, Gulfstream Group and bigbyte.cc, Intel Corporation, Los Alamos National Security, LLC, One Connect IP, Technology Integration Group, and ZiaNet.

Bronze partners are Apogentech, Albuquerque Journal, BX Internet, Cray Inc., Lobo Internet Services, New Mexico Business Weekly, New Mexico Technology and Council,

Redfish Group, Jim Stewart, and Strategic Analytics, are Sun Microsystems “Bronze” commercial partners.

Los Alamos National Laboratory, a multidisciplinary research institution engaged in strategic science on behalf of national security, is operated by Los Alamos National Security, LLC, a team composed of Bechtel National, the University of California, The Babcock & Wilcox Company, and URS for the Department of Energy’s National Nuclear Security Administration.

Los Alamos enhances national security by ensuring the safety and reliability of the U.S. nuclear stockpile, developing technologies to reduce threats from weapons of mass destruction, and solving problems related to energy, environment, infrastructure, health, and global security concerns.

More information on the New Mexico Supercomputing Challenge can be found at <http://www.challenge.nm.org> online, while final student reports are available at <http://www.challenge.nm.org/archive/09-10/finalreports> online.

Teams Finishing the Challenge and submitting final reports:

Team 3, Albuquerque Institute for Math and Science, *Ant Recruitment*

Team Members: Nico Ponder, Stefan Klosterman, Jordan Medlock, Erik Johnson
Sponsor: Terrence Lebeck, Mentors: Mark Johnson, Tatiana Paz

Team 5, Albuquerque Academy, *The Metropolis Algorithm and Nanometer-Scale Pattern Formation*

Team Members: Michael Wang, Jack Ingalls
Sponsor: Jim Mims, Mentor: David Dunlap, Ph.D

Team 11, Artesia High School, *Sugarscaping On a Beowulf Ring*

Team Members: Wesley Green, Isaiah Jordan, James McGee, Wen Hai Zheng
Sponsor: Randall Gaylor, Mentors: Nick Bennett, Jose Quiroz

Team 13, Artesia High School, *Classroom Behavior*

Team Members: Cristina Villa, Nayeli Ramirez, Brenna Arredondo
Sponsor: Amy Mathis, Mentors: Randall Gaylor, Nick Bennett, Olivia Rueda

Team 15, Aspen Elementary, *How Not to Become a Global Pandemic Statistic*

Team Members: Pippa Chadwick, Claire DeCroix, Evan Oro, Claire Ticknor
Sponsor: Zeynep Unal, Mentors: David DeCroix, David Oro

Team 16, Aspen Elementary, *Whose Fault Is It?*

Team Members: Talia Dreicer, Hunter Eaton, David Smith
Sponsor: Zeynep Unal, Mentors: Jared Dreicer, Kathy Smith

Team 17, Aspen Elementary, *Powering Los Alamos With Solar and Wind Energy*
Team Members: Emma Martens, Rachel Wallstrom
Sponsor: Zeynep Unal

Team 18, Aspen Elementary, *The Disappearing Honeybees*
Team Members: Kim Vo, Rowan Cantua, Kaelan Prime
Sponsor: Zeynep Unal, Mentors: Duc Vo

Team 20, Bloomfield High School, *Adaptive Virus*
Team Members: Cameron Corley, Matthew Crockett, Jon Romero
Sponsor: Elvira Crockett

Team 21, Bloomfield High School, *Evolution of Influenza Over Time*
Team Members: Jonathan Zamora, Simone Valdez, Alex Jim, Evelyn Gutierrez
Sponsor: Elvira Crockett, Mentor: Elvira Crockett

Team 23, Creative Education Preparatory Institute #1, *Firework Hearing Loss*
Team Members: Arlene Pino, Angela Caudle, Chance Lammey
Sponsors: Jerry Esquivel, James Stewart, Mentors: Mark Murmer, Judith Velarde, Laura Rowen

Team 24, Creative Education Preparatory Institute #1, *The Virus*
Team Members: Michael Szanto, Devin Hayes, Ryan Fitzgerald
Sponsors: Jerry Esquivel, James Stewart

Team 25, Creative Education Preparatory Institute #1, *Neurology and Epilepsy*
Team Members: Sara “Katelynn” Higgins, Carlos E. Reazin
Sponsors: Jerry Esquivel, James Stewart, Mentor: Dr. Shiboya

Team 27, Creative Education Preparatory Institute #1, *The Last Virus*
Team Members: Robert Lopez, Robert Parnell
Sponsors: Jerry Esquivel, James Stewart

Team 29, V. Sue Cleveland High School, *The Percentage of Disease During a Common School Day*
Team Members: Louis Taylor, Ben Fowler
Sponsor: Debra Loftin, Mentor: Nick Bennett

Team 30, V. Sue Cleveland High School, *Accelerated Particles vs. Metastatic Cells*
Team Members: Matthew Bradly, Jeremy Wright, Ryan Cortez, Kevin Clay
Sponsor: Debra Loftin, Mentor: Nick Bennett

Team 33, Deming High School, *Tortuga Trouble: A New Survey Method*
Team Members: Rocky Navarrete, Gabriela Anguiano
Mentor: Creighton Edington

Team 35, Deming High School, *Help Me, Doctor!*
Team Members: Amanda Edington, Erick Chávez
Sponsor: Creighton Edington

Team 36, Desert Academy, *Arbitrary Precision Integers on the Cell Processor*
Team Members: Megan Belzner, Matt Rohr, Bjorn Swenson
Sponsor/Mentor: Thomas Christie

Team 37, Desert Academy, *The Spread of the Black Death in London*
Team Members: Katie Boot, Sara Hartse
Sponsors: Thomas Christie, Jocelyn Comstock

Team 38, Desert Academy, *Socialist Manifesto*
Team Members: Isaac Green, Sean Collin-Ellerin
Sponsor: Thomas Christie

Team 39, Edgewood Elementary School, *Red Hot Chili Peppers*
Team Members: Natasha Cordova, Deyvy Armendariz, Olivia Riblett
Sponsors: Jennifer Cordova, Carol Thompson, Mentors: Ryan Serrano, David R. Janecky

Team 40, Edgewood Elementary and Middle School, *When Pigs Fly*
Team Members: Timothy Thompson, Joshua Berson, Casey Bond, Joseph Shaffier
Sponsor: Carol Thompson, Mentor: Christopher Hoppe

Team 48, Freedom High School, *The Rising Socorro Magma Body*
Team Members: Joel Sandoval, Marika Plugge, Yoshua Reece, Jasmine Jensen
Sponsor: Joe Vertrees, Mentor: Paula Rimmer

Team 50, Freedom High School, *A Proper Interpretation of Panic*
Team Members: William Barrett, Holly Campbell, Chelsea Kibbee, Amy Ronan

Team 51, Hatch Valley High, *Prediction of Green Chile*
Team Members: Yoliy Gamboa, Joel Cazares
Mentor: Creighton Edington

Team 53, Los Alamos Homeschool, *Grocery Tracker*
Team Members: Ariel Koh, Aline Parlman
Mentor: Dr. Aik-Siong Koh

Team 56, Los Alamos Homeschool, *Get on the Bus 2*
Team Members: Isaac Koh
Sponsor/Mentor: Aik-Siong Koh

Team 57, Hope Christian School, *Heart Attack*
Team Members: Alexander Alvarez, Angela Wise, Elysia Berg
Sponsors: Pam Feather, Sue King, Mentor: Pam Feather

Team 59, Hope Christian School, *Learning in Space*
Team Members: Cameron Harjes, Aaron Gabaldon, Tyler Spahr, Ben Robinson
Sponsor: Sue King, Mentor: DeLesley Hutchins

Team 60, Hope Christian School, *Video Games, the Moral Decline in America*
Team Members: Jonathon Kruse, Alex Jennings, Burke Wilson
Sponsors: Pam Feather, Sue King

Team 61, Jackson Middle School, *Undercover Bruise*
Team Members: Karina Ortega, Kelsey Theriot, Sandra LeNguyen
Sponsor: Karen Glennon, Mentor: Nick Bennett

Team 62, Jackson Middle School, *Jellyfish Domination*
Team Members: Brendyn Toersbijns, Thomas Hughey, Spenser Gomez-Nelson
Sponsor: Karen Glennon, Mentor: Nick Bennett

Team 64, Jackson Middle School, *Water Purification*
Team Members: Christopher Hoebing
Sponsor: Karen Glennon, Mentors: Nick Bennett, Betsy Frederick

Team 65, Los Alamos High School, *The Holy Grail of Adam's Ale*
Team Members: Gabriel Montoya, Rachel Robey, Orli Shlachter, Orion Staples
Sponsor: Lee Goodwin, Mentors: Robert Robey, Thomas Robey

Team 67, Los Alamos High School,
*Save Energy, Part 1: Numerical Method for Heat Conduction In Systems of Arbitrarily
Different Materials*
Team Members: Edward Dai, Aidan Bradbury
Sponsor: Lee Goodwin, Mentor: William Dai

Team 68, Los Alamos High School, *Astrophysical N-Body Simulations of Star Clusters*
Team Members: Sam Baty, Peter Armijo
Sponsors: Lee Goodwin, Diane Medford, Mentors: Roy Baty, John Armijo

Team 69, Los Alamos High School,
Enlightenment: Metropolis-Hastings Ray Prediction Model in 3D Space
Team Members: Kathy Lin, Jake Poston, Ryan Marcus, Dov Shlachter
Mentors: Leroy Goodwin

Team 70, Los Alamos High School, *To Kill a Flocking Bird*
Team Members: Peter Ahrens, Stephanie Djidjev, Vicky Wang, Mei Liu
Sponsor: Lee Goodwin, Mentors: Christine Ahrens, James Ahrens

Team 73, Los Alamos Middle School, *Artificial Intelligence used in a Battle Simulation*
Team Members: George Barnum, Mohit Dubey, Ben Liu
Sponsor: Clara Vigil, Mentor: Bob Robey

Team 75, Los Alamos Middle School, *Smart Grid*
Team Members: Colin Redman, Michael Englert Erickson, Sudeep Dasari
Sponsor: Clara Vigil, Mentors: Andrew Erickson, Jim Redman, Venkat Dasari

Team 79, Los Alamos Middle School, *Alien River Cloggers*
Team Members: Jacob Holesinger, Kevin Tao
Sponsor: Clara Vigil, Mentor: Terry Holesinger

Team 81, Manzano High School, *Java-Based Wireless Robot*
Team Members: Philip Atencio, Dustin Chavez, Nathan Hassler, Nick Ratzler, David Young
Sponsor/Mentor: Stephen Schum

Team 82, Manzano High School, *Remote Combined Solar and Wind Renewable Energy Power Grid*
Team Members: Hathaweh Bassett, Hung Nguyen
Sponsor: Steve Schum

Team 83, McCurdy High School, *Bridge Destruction*
Team Members: Carlos Herrera, Louis Jaramillo, Justin Garcia, Ron DeVargas
Sponsor/Mentor: Robert Dryja

Team 84, McCurdy High School, *Particle-Atmosphere Interaction*
Team Members: Dennis Trujillo, Oliver Galvan, Brandon Ricci
Mentors: John Pretz, Brenda Dingus, Philip Sanchez

Team 85, McCurdy High School, *Contributing Factors for Obesity in the U.S.*
Team Members: Lindsay Redman, Isabel Garcia, Marisa Griego
Sponsor: Irina Cislaru, Mentors: Jennifer Tichy, Alin Panaitescu

Team 97, Melrose High School, *Polar Ecosystem*
Team Members: Quinton Flores, Brian Hemminger, Kira Anderson
Sponsors: Alan Daugherty, Rebecca Raulie

Team 98, Melrose High School, *Surviving the Worst*
Team Members: Brandon Mitchell, Victoria Northrup, Adrianna Saiz, Allicyn Trammell
Sponsors: Alan Daugherty, Rebecca Raulie

Team 99, Melrose High School, *Control and Spread of Wildfires II*
Team Members: Richard Rush, Kyle Jacobs, Randall Rush
Sponsors: Alan Daugherty, Rebecca Raulie

Team 102, Navajo Preparatory School, *The Latest Buzz About Bees*
Team Members: Alexis Archambault, Ariel Nephew, Wilfred Jumbo
Sponsor/Mentor: Mavis Yazzie

Team 103, Navajo Preparatory School, *What's Up With the Ozone Layer*
Team Members: Malcolm Bob, Leland Gray, Malcolm Keetso
Sponsor/Mentor: Mavis Yazzie

Team 104, Northern New Mexico College,
New Mexico on the Road: Impact of Fuel Consumption and CO2 from NM Cars
Team Members: Kaitlyn Dow, Annaleah Dow, Jeremy Salazar, Angela Gomez
Sponsor: Jorge Crichigno

Team 107, Quemado High School & Silver High School, *First Impressions*
Team Members: Jose Mora, Austin Nightengale
Sponsors: Laura Larisch, Peggy Lerisch

Team 108, Quemado High School, *Can You Hear Me Now?*
Team Members: Janessa Larrabee, Judith Flores
Sponsors/Mentors: Laura Larisch, Peggy Larisch

Team 109, Quemado High School, *Oil Spills in Oceans*
Team Members: Justin Miller
Mentor: Laura Larisch

Team 110, Red Mountain Middle School, *Marble Roller Coaster "Thrill Ride"*
Team Members: Bryce Golie

Team 112, Sandia Preparatory School, *Analytical Hierarchal Process for Complex Decisions*
Team Members: Caitlyn Scharmer
Sponsor: Neil McBeth, Mentors: Minga Banks, Carol Scharmer

Team 119, Silver High School, *Airplane Epidemiology*
Team Members: Forest Brown
Sponsor: Peggy Larisch, Mentor: Dr. Camacho

Team 125, Tibbetts Jr High, *Recycle This*
Team Members: Jesse Duarte, Jacob Hensley
Sponsor: Ms. Maurer, Mentor: Bob Robey

Team 127, Manzano High School, *Efficient Air Traffic Control*
Team Members: Ryan Hensel, Elisabeth Keller, Jelke Adema
Sponsor: Steve Schum

Judges

Toru Aida, Los Alamos National Laboratory
Dr. Ed Angel, Santa Fe Complex
Dorian Arnold, University of New Mexico
Dorothy Ashmore, Sandia National Laboratories
Richard Barrett, Sandia National Laboratories
Nick Bennett, Grass Roots Consulting
Louise Byrd, APS Gifted Program, retired
Dr. Patrick Bridges, University of New Mexico
Jon Brown, New Mexico Tech
Powell Brown, New Mexico Tech
Kent Budge, Los Alamos National Laboratory
Cheri Burch, University of New Mexico
Dr. Chuck Burch, University of New Mexico
Dr. Jorge Crichigno, Northern New Mexico College
Roger Critchlow, File Systems Lab
Dr. Shaun Cooper, New Mexico State University
Dr. Larry Cox, Los Alamos National Laboratory
Mike Davis, Cray
Erika DeBenedictis, Albuquerque Academy
Dr. Sharon Deland, Sandia National Laboratories
Drew Einhorn, Fourth Watch Software
Dr. Gary Geernaert, Los Alamos National Laboratory
Dr. Bill Godwin, Lockheed-Martin Fellow, retired
John Paul Gonzales, Santa Fe Institute
Lisa Harris, Los Alamos National Laboratory
Willard Hemsing, Los Alamos National Laboratory
Harry Henderson, Rio Rancho Schools
David Janecky, Los Alamos National Laboratory
Dr. Philip Jones, Los Alamos National Laboratory
Sharad Kelkar, Los Alamos National Laboratory
Larry Kilham
Victor Kuhns, Cray
Larry Landis, Fourth Watch Software
Dr. Tom Laub, Sandia National Laboratories
Dr. Maximo Lazo, Science Applications International Corporation
Irene Lee, Santa Fe Institute/Project GUTS
Dr. Lorie Liebrock, New Mexico Institute of Mining and Technology
Debbie Limback, San Juan College
Jonathan Margulies, Sandia National Laboratories
Nico Marrero, New Mexico Tech
Dr. Cleve Moler, The MathWorks
Lonny Montoya, New Mexico Highlands University
Dr. James Overfelt, Sandia National Laboratories
Kathy Pallis, Los Alamos National Laboratory

Georgia Pedicini, Los Alamos National Laboratory
Dana Roberson, Department of Energy
Doug Roberts, RTI International
Dr. Tom Robey, Gaia Environmental Sciences
Bob Robey, Los Alamos National Laboratory
Janet Rolsma, Department of Energy
Shawn Shay, San Juan College
Dr. Willard Smith, Tennessee State University
Jack Stafurik, Technology Ventures Corporation
Julianne Stidham, Los Alamos National Laboratory
Mike Trahan, Sandia National Laboratories
Dr. Eleanor Walther, Sandia National Laboratories
Tim Warren, San Juan College
Dr. Suzanne Westbrook, University of Arizona
Talaya White
Beryl Wootton, New Mexico Tech
Peter Yanke, BX Internet

Do you want to become a supporter of the Supercomputing Challenge? Please email us at consult@challenge.nm.org for details.

CONTROL AND SPREAD OF WILDFIRES II

New Mexico

Supercomputing Challenge

Final Report

September 14, 2010

Team: 99

Melrose High School

Team Members:

Richard Rush

Kyle Jacobs

Randall Rush

Teachers:

Allan Daughtery

Rebecca Raulie

Table of Contents

Problem Explanation.....	3
Solving the Problem.....	3
Mathematical Model	6
Last Year’s Results	9
Our Goals	10
This Year’s Results	10
Firemen	10
Firebreaks.....	11
Distractions	15
Fire Fighting Techniques	16
Executive Summary	16
Bibliography	18
Acknowledgements.....	18

Problem Explanation

This year we are expanding upon last year's project on the control and spread of wildfires. Fire has been a very important part of our lives and will continue to be so in the future, in both positive and negative ways. Wildfires cause extensive amounts of life and property damage; and we wanted to know the best way to extinguish a fire in progress and the best way to protect property from an out of control blaze.

Last year we created a model of a fire on a flat plain. We included the variables of fuel load, wind speed, wind direction, and moisture content. We then took our model to the local fire department and were able to accurately model a fire that they recently fought, verifying the reliability of our model.

This year, we have added a third dimension to our model, topography. On the Llano Estacado this variable did not affect our model much, however, in most other parts of the world it can be the most prominent factor affecting the fire's behavior. We then had to develop a procedure to cause the wind to react to the topography. Firefighters who try to fight the fire and protect a residential area were also incorporated into the model. Then we tested various firebreaks to determine which design is the most effective in diverting or slowing a fire's progress.

Solving the Problem

Last year we began our model in "StarLogo TNG," an agent based model. We did have plans to expand our modeling this year into "NetLogo" (a more advanced and versatile modeling program), however due to time constraints and limiting our project to goals to those that we could achieve in one year, we simply expanded our old model. This was, as we found out, not a detriment to this year's project goals. With StarLogo TNG, we can set up different types of fire breaks easier and more quickly than we could with NetLogo, as we understand the language.

We solve the various problems by introducing an agent in "SpaceLand" (The area where the agents will operate) that will consume fuel and multiply and progress in accordance to the following variables:

Fuel Load—The amount of fuel available per square area of land. This represents different vegetation types and densities. This will be represented in our model by a scale of 1-10 in shades of green. The darker the green, the greater the fuel load. The fuel load in an area can change greatly, on one side of the fence there can be a green wheat field which has an extremely low fuel load, and on the other side of the fence, there can be land enrolled in CRP (Conservation Reserve Program) which can have a very high fuel load. The fuel load in a single pasture can change too, depending on what kind of grass grows from place to place and the quality of the soil across a field. The fuel loads that we will use for our model will be:

- 1) Very, very low—less than 200 pounds of fuel per acre
- 2) Very low—from 200-500 pounds of fuel per acre
- 3) Moderately low—from 500-800 pounds of fuel per acre
- 4) Moderate—from 800-1100 pounds of fuel per acre
- 5) Moderately high—from 1100-1500 pounds of fuel per acre
- 6) Very high—over 1500 pounds of fuel per acre

Moisture Content—The amount of moisture in the area. Often the land will be very dry before a thunder storm, therefore, lightning can easily start a fire. As the storm progresses it may rain, increasing the moisture, retarding the fire's progress, and possibly even putting the fire out. This variable can also represent the growing stage of the vegetation represented.

Wind Direction—Wind direction plays a vital role in fire behavior and control. If there is a wild fire, the fire fighters may have had it under control or could have had a firebreak set up ahead of it, but if the fire suddenly changes direction, it may become uncontrolled once again.

Wind Speed—Wind speed is also very important in fires. A fire under a light wind might not ordinarily cross a firebreak or obstacle such as a road, but if the wind were blowing

enough, the road would barely slow down a raging fire. We will use these numbers to represent the different wind speeds:

0—0-3 mph

3—15-25 mph

1—3-10 mph

4—over 25 mph

2—10-15 mph

Topography—The heat that radiates from a fire tends to rise, therefore a fire will advance much more quickly up a hill than down a hill. The wind will also be affected by the terrain. A hill can provide a wind break on one side and channel wind down a canyon on the other side of the hill, thus changing wind speeds and directions. We control this variable with “StarLogo TNG’s” versatile terrain editor.

With control of these variables in our model, we will “start a fire” on the map and let it burn until the entire map is consumed or the fire burns itself out. We can then alter the map by adding a firebreak and running the model again.

Our model runs on these basic principles:

1) Burn

a) Test to see if patch color is some shade of green

b) If so, then add one shade of white to the patch.

c) If not, then test to see if patch is already burned

i) If so, then have a 66% chance of dying (This gives a fire a chance at crossing a firebreak with sparks and tumbleweeds).

ii) If not, then have a 66% chance of dying.

(The 66% was determined through trial and error relying on local firefighter’s experience of fire behavior.)

2) Spread

- a) Test to see if a random number between 0-100 is less than or equal to dryness.
- b) If so, then choose a random number between 0-360 for a direction
 - i) Then create a new fire and send it 1 step in the chosen direction.

3) Wind

- a) Select a random number between 0-45 and add it to the wind direction
- b) Then divide wind speed by 2 and add 0.5 and take that many steps in the chosen direction.

4) Hills

- a) Test if patch height is less than patch height ahead.
 - i) If so, then go forward the difference between patch height and patch height ahead times 4.
 - ii) Hatch
- b) Test if patch height is greater than patch height ahead.
 - i) If so, then go back the difference between the patch height ahead and patch height times 1.
 - ii) Have a 50% chance of dying.

Mathematical Model

The mathematical formulas that our model follows are stated below.

Setup

F_1 = first fire agent generated

F_1 is randomly placed on an x y grid according to ...

$$x \sim U[-50.5, 50.5)$$

$$y \sim U[-50.5, 50.5)$$

Where U is uniform distribution.

F_1 is randomly placed on a “patch”.

A “patch” is an area that is centered on an x y grid where $x_0, y_0 \in \mathbb{Z}$

A “patch” includes...

$$x \in [x_0 - 0.5, x_0 + 0.5)$$

$$y \in [y_0 - 0.5, y_0 + 0.5)$$

Spread

f_i = fire agent i

t = time

P = probability

The probability that a fire sparks another fire is ...

$$P(f_i \text{ produces } f_j), [t, t+1)$$

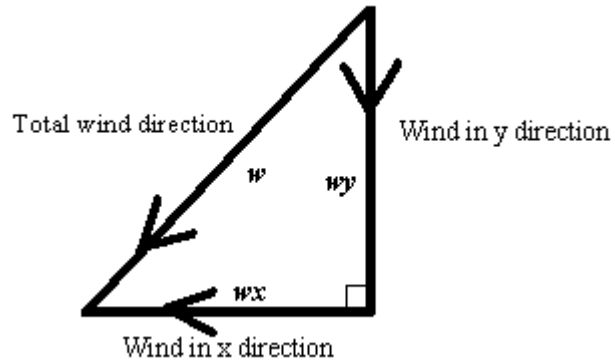
The location of f_j is...

$$x_j = x_i + \cos \theta r + w \mathbf{x}$$

$$y_j = y_i + \sin \theta_r + w_y$$

Where “ θ_r ” is a random angle uniformly distributed on $[0, 2\pi)$

Where “ w ” is a wind vector composed of x and y directions



Burn

L_k = fuel level in “patch” K

The fuel level decreases by the number of “fires” on that “patch”

$$L_{kt+1} = \max(L_{kt} - n, 0)$$

Where “n” is the number of fire on “patch K”

If $L_{kt} = 0$, all agents on “patch K” “die”, in the time interval, $(t, t+1]$

Hills

If there is a positive grade

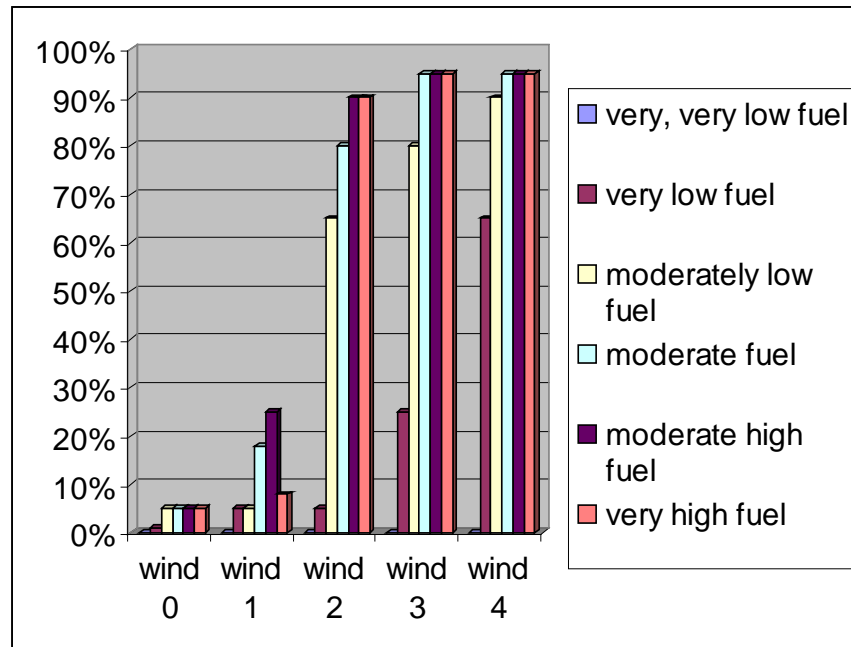
F_I moves forward (marginal difference)(4)

If there is a negative grade

F_I moves backwards (marginal difference)(1)

Last Year's Results

Last year we were able to accurately model a fire and did a lot of work with roads and how the fire reacted to them. We used county roads as our standard. We started with the fire approaching a road at a perpendicular angle. We tested this under varying fuel loads and wind speeds and directions. The following graph shows the percent chance that the fire has of crossing the road under the specific conditions.



We did not give anything a 0% chance or a 100% chance as there is a lot of random numbers and anything could happen if the model were run enough times.

We also tested the impact of varying fuel loads in drainage ditches that usually run next to roads. They can either have higher fuel, due to being able to receive larger amounts of runoff, or they can have a lower fuel load from the state or county mowing it off. We found that under conditions in which a fire didn't usually cross, the fire was able to cross the road if the ditch contained a high fuel load. On the other hand if the ditch had a lower fuel load, it did delay the fire's crossing significantly.

Another factor that we tested, was whether or not the fire's angle of approach to the road affected its chance of crossing the road. We found that the lower the fire's angle in comparison

to the road, the less chance it had of crossing over the road. This was due to the fire traveling with the road instead of against it. We used this information in the development of our firebreaks.

Our Goals

This year we decided to use the model that we created last year to find the best ways to slow, stop, or divert a fire with firebreaks, as well as to find the most effective way for the firemen to work with the firebreak to get the fire under control and extinguished thereby saving lives and property. With StarLogo TNG, it is easy to manipulate the terrain for different scenarios, and set up different types of firebreaks for testing.

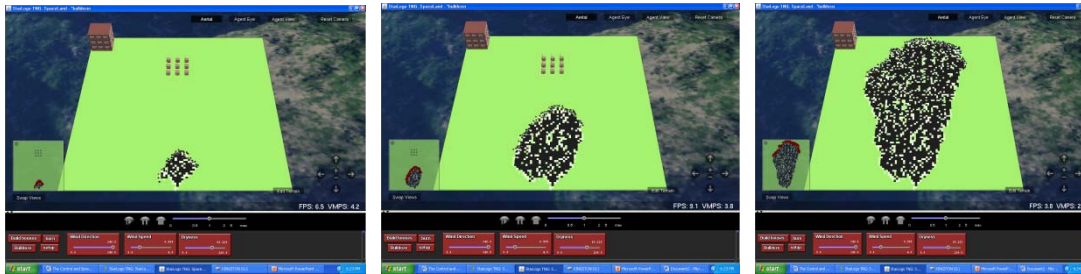
This Year's Results

Last year we developed a model that accurately portrays how a fire will act under certain variables and verified that our model is viable. This year we modeled various situations and fire breaks so that we would be able to see how the fire would act under the given situation. We tested two fuel loads in the different firebreak designs. We determined firefighter's ability with each fire break design to protect property by counting how many houses were left at the end of each test. We further pursued the fire's advancing angle to the firebreak.

Fire Men

This year we added firefighters to our model. We set up independent agents to search for and put out the fire. Currently we have a few firemen with fire extinguishers killing individual fire agents when there can possibly be thousands of fire agents so, our fire fighters can easily be overwhelmed. However, this is a great step towards setting up a program that would test firefighting methods previously impractical due to safety hazards. This is also getting us closer to another important piece of information, how much water will it take to put the fire out? As fire trucks can only carry a limited amount of water it would be a great benefit for the fire fighters to know how much water or fire fighting agent is needed to extinguish a fire under various conditions.

Right now though, our firemen aspect of the model shows how important early fire control is and how little effort is needed to put a small fire out before it gets out of control. In this model we have a village placed in the way of a fire. The wind is about 10 mph with damp conditions and a low fuel load.



As you can see, the fire was able to build up and envelope the village. In the next model we let 10 firemen find the fire and proceed to try to put it out.



In this model the firemen were able to find the fire and put it out before it had a chance to grow. The firemen were able to save the village from the fire.

We have also found that if an area is in danger, it works best if the firefighters “stage” in front of it, however, they can’t be too close or too far away, because if they are too close they do not have time to successfully engage the fire. However, if they are too far away, the firemen’s resources are too dispersed to protect the village if the fire gets by them.

Fire Breaks

We tested several different types of firebreaks to learn which one would be the most effective in diverting a fire or slowing it. We tested various shapes and fuel loads for the

firebreaks, as well as fire direction in relation to the position of the firebreak. We also tested whether the break slowed the fire enough for the firemen to extinguish it. We measured the firebreak's effectiveness by placing nine houses in front of the fire to represent a village, and recorded how many houses survived the fire.

One of the things that we took into account when we designed the firebreaks was feasibility and aesthetics. For example, you could create a very effective firebreak for your house by plowing up all the vegetation and converting it to mineral soil all around your property for one hundred yards in all directions. This would pose a few problems for the people living in that residence. It would probably be very dusty and the plowed up area would be subject to wind erosion, removing the topsoil from the area. It would not look very good to have a huge brown square all around your house either. So, we tried to keep the fire breaks to areas of low fuel or small areas of no fuel.

To analyze our results more accurately we used Microsoft Excel's statistical functions. Since we only ran each model 10 times, we used small sample inference and the Student's t test (which our mentor found appropriate). The equation for Student's t is as follows:

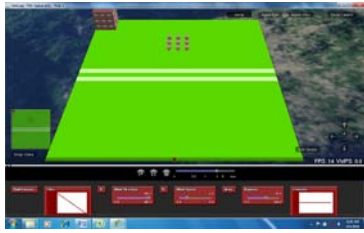
$$t = \frac{\bar{Y} - \mu}{S/\sqrt{n}} = \frac{\bar{Y} - \mu}{S_{\bar{Y}}}$$

We used the Student's t to determine how different our data sets were. In this case how the firebreak's effectiveness compared to a control of no firebreak. We also used this function to determine if one design was significantly better than others.

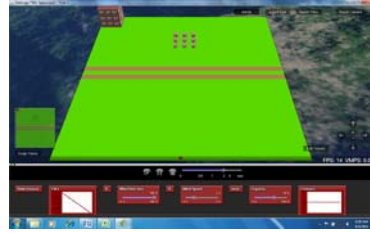
We tested the following fire breaks under these conditions. The moisture content was set at 40%, the wind speed was at a 1 or between 3 and 10 mph, the wind direction was blowing the fire directly at the village, the base fuel load was moderate, the village consisted of nine houses arranged in a block, and firefighters were extinguishing the fire. We started with no firebreak and then added different designs: two strips of no fuel, two strips of low fuel, a solid strip of low fuel, and an arrow or V shape around the village. However, StarLogo TNG does not allow us to draw a homogenous diagonal line. This reduced the effectiveness of our angled breaks.



Control, no firebreak



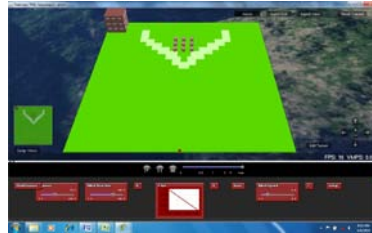
Two Lines, low fuel



Two Lines, no fuel

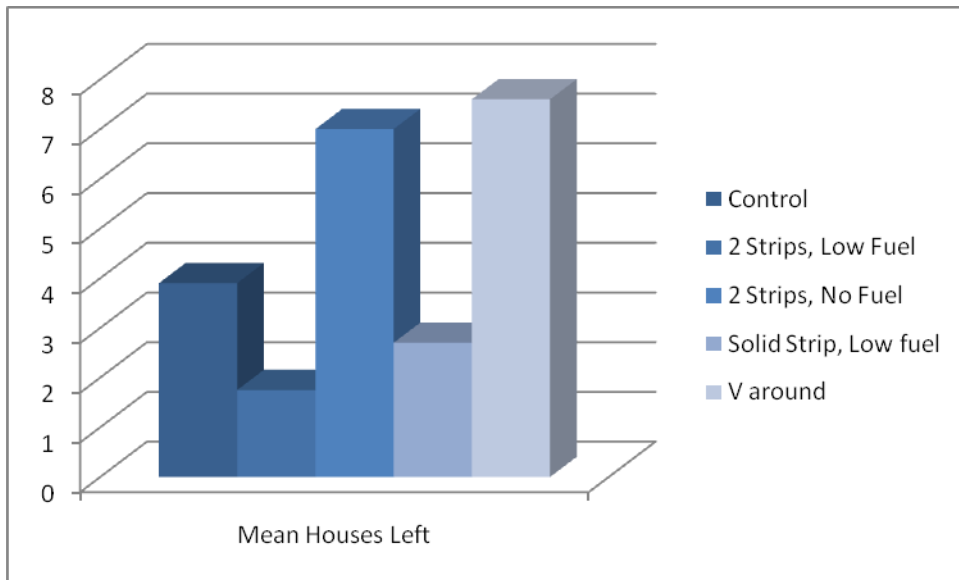


Single Line, low fuel



V design, low fuel

The following graph shows the average number of houses that survived the fire after running each model 10 times.



From the information presented in the graph we were able to infer a couple of things. First of all, we were surprised to find that not all firebreaks were helpful. Next we found that a firebreak that was more effectively designed with low fuel was better than a straight firebreak with no fuel. When we noticed the first problem, we ran our model more times and examined the

behavior more closely. The reasons the firebreaks seemed to be detrimental was that the firebreak was actually doing its job, it was slowing the fire down as indicated by our fire population graph, but it was also widening the head of the fire spreading the firefighters resources out too thinly to stop the fire's progression.

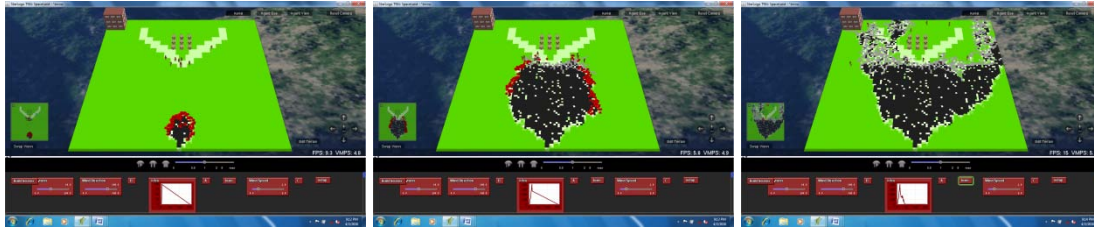
To more accurately determine the differences between the breaks we have run a t-test on the above sets of data testing the control against the firebreaks and testing the best firebreak against the others. The following table shows the result of the t-test.

Firebreak	control	2 Strips, Low Fuel	2 Strips, No Fuel	Solid Strip, Low Fuel	V Around
T-test	1	0.13	0.06	0.45	0.02
(Rounded to 2 decimal places)	0.02	0.00	0.60	0.00	1

These numbers show that the V around and the two strips of no fuel were significantly better than the control. The test also showed us that the V around is not significantly better than the two strips of no fuel. There is strong evidence, 87%, that the two strips of low fuel were detrimental to the survival of the town. Since we were very surprised by these results, we doubled our sample size and nothing changed.

We can also deduct that an effectively designed and placed firebreak is better than just an ordinary straight line. From the graph we can tell that the V design saved, on average, more houses than the next best firebreak, the two lines with no fuel. From the statistical analysis, however, we see that there is only a 40% chance that it is a significantly better firebreak. These chances are not good enough for us to draw an immediate conclusion (for us to do so we will have to do a significant amount of more testing), but right now, we will give the upper hand to the V break due to erosion problems and aesthetics mentioned earlier.

Here, we have the fire break that performed the most desirably, the V, or arrow shaped pointing in the direction that the fire is coming from, and surrounds the town. It has moderate fuel load, with the break made of very low fuel.



The fire men staged just to the houses side of the fire break, and were able to stop the fire, and save all of the houses. We then ran this model by placing the houses on the top of a hill and by placing the houses in the bottom of a low spot with and without the V break. In both cases we found that a firebreak was better than no firebreak.

The firebreak's angle to the direction of the fire was also tested. However, due to limitations in the program, we were not able to model this as accurately as we wished we could have. But we did strengthen several beliefs about the fire's behavior. First of all, the fire has more difficulty in crossing a break when it is running at an angle to it or with it. This helps explain why diagonally shaped fire breaks work the best at diverting a fire away from the village. An area with no fuel is a better break than an area with low fuel. Firefighters are better extinguishing agents than simply a firebreak. The firefighters need to effectively stage the fire or use the firebreak to their best advantage. Such as, the firebreak is not as helpful if the firefighters go out in front of it, because if the fire gets past them, then they are stuck behind the fire. The firebreak is also not as effective if the firefighters are too far behind it, because it has had time to increase to its original intensity.

Distractions

While working with our firemen and firebreaks we have discovered a very interesting result of having them together. In one model the firemen were to stay behind the firebreak and let the firebreak slow the fire down so it was easier to put out. However, when the fire crossed the break in one place, all of the firemen ran to put it out. While the firemen "had their backs turned," you could say, the fire would cross the break behind them and would destroy the village before the firemen could do anything about it. This usually occurred with a relatively strong fire break as well, and we were very surprised to find that it did not work as well with the firefighters as we anticipated. We have heard that at some real fires all of the firemen would run up to the

largest part of the fire to put it out, while, unbeknownst to them, the fire is threatening a residence elsewhere. This shows how it is important to have a chain of command in the field with someone directing the firemen and trucks.

Firefighting Techniques

While testing firebreaks we also discovered some firefighting tactics that are effective in diverting the fire away from residential areas or other locations of interest. These can change according to from which direction the fire is approaching. When the fire is coming straight at a village it can be more effective to wait for the fire to get close to the village, then concentrate all firefighting resources at one point in front of the fire. This will split the fire in two and allow the firefighters to work on the inside flank of the fire and push it around the village.

However, if the fire is approaching at an angle to the village, we would suggest that the firefighters stage near the village on one flank of the fire; this would push the fire around one side of the village. The firefighters could then work on putting the rest of the fire out when the village is safe.

Executive Summary

This year, we improved upon last year's wildfire model by inputting topography, which is very important for some areas. We were able to get the wind to react to the topography, because wind can funnel through canyons, and hills or mountains can act as wind breaks. This was the most difficult to simulate. We also implemented firemen, one of the most crucial resources that a community has in protecting lives and property.

We used our more advanced model to test different varieties of firebreaks and fire fighting tactics. We found that out of the firebreaks that we tested, a V-shaped firebreak with the village inside of it was the most effective. We also found through further testing, that firebreaks are more effective if they are placed at an angle to the fire's advancing direction. The firemen perform best if they stage just behind the firebreak, and put it out as it is trying to cross. Not all firebreaks are helpful; the firebreak has to influence the fire in such a way as to give the firemen a strategic advantage. Firefighters in the field need a chain of command and a common goal to help organize them so they will be able to use their resources more effectively.

By expanding our model this year, we have tested the limits of StarLogo TNG. It has been a very effective tool in verifying our procedures and models. However, it is now starting to limit us. We cannot import real topographical maps from outside sources, we have a limited amount of space in SpaceLand, and we cannot build custom procedures. There is also a limit to the number of agents that can be in SpaceLand at once. This may be a moot point since we seem to have reached the limits of our PC. Sometimes so many agents were running so many procedures, that the program would become confused and would either shut off entirely or mix up breeds of agents and the procedures that they were supposed to follow. To continue this project we need to change our modeling program to enable us to overcome these obstacles and to eventually meet the goal of creating a marketable program to be used by fire departments, cities, and individuals to develop more effective firefighting techniques or just how to most effectively protect their own home by custom designing firebreaks to meet an individual's needs.

Bibliography

Holecheck, Pieper, Herbel, Range Management, Principles and Practices,. 3.

"Fire." wikipedia the free encyclopedia. 2008. 20 Nov 2008

"Fire." FEMA.gov. 2006. 30 Mar 2009

Harris , Tom. "How Fire Works." HowStuffWorks.com. 1998. 20 Nov 2008

David Rush, Forrest Volunteer Fire Department, Training Coordinator

Addison-Wesley, Chemistry of Firefighting, copyright 1990

Bender, Douglas, Kramer, Statistical Methods for Food and Agriculture, copyright 1982

Acknowledgements

We would like to thank Nick Bennett for coming all the way to Melrose from Albuquerque to help us with our programming and developing our mathematical model.

We would like to thank John Paul Gonzales for showing us around Santa Fe at Swarmfest, for listening to our presentation multiple times, and for coming to Melrose to help us with our programming.

We would also like to thank David Rush of the Forrest Fire Department for helping us determine how to most accurately set up our model.

The Holy Grail of Adam's Ale
Locating Aquifers through Geostatistic Modeling

Team #65

April 7, 2010

New Mexico Supercomputing Challenge Final Report

Los Alamos High School

Team Members:

Gabriel Montoya
Rachel Robey
Orli Shlachter
Orion Staples

Teacher Sponsor:

Lee Goodwin

Mentors:

Robert Robey
Thomas Robey

Contents

1	Introduction	6
1.1	Problem Statement	6
1.2	Objective	6
1.3	Background	6
1.3.1	Geostatistics	6
1.3.2	Aquifers	7
2	Mathematical Models	8
2.1	Semi-variogram	8
2.2	Interpolation	9
2.2.1	Inverse Distance Weighting	10
2.2.2	Kriging	10
2.3	Sampling from Gaussian Distribution	12
3	Computational Model	13
3.1	Semi-variogram	13
3.2	Solution of Kriging Equations	14
3.3	Anisotropy	17
3.4	Sampling from Gaussian Distribution	18
3.5	Multiple Points and Runs	18
3.6	Optimization	19
3.6.1	Variation on Random Iteration Algorithm	19
3.6.2	Transferring Matrix Solver to the GPU	20
4	Code	23
4.1	Overview and Structure	23
4.2	WxWidget Windowing	24
4.3	Computational	26
4.4	OpenGL Graphics	26
4.5	wxPlotCtrl Graphing	28
5	Results	29
5.1	Case Study	29
5.1.1	Data	29
5.1.2	Semi-variogram	31
5.1.3	Mixed Success of Interpolated Fields	32
6	Conclusions	33
6.1	Current Status	34
7	Teamwork	34
8	Recommendations	35

A	References	35
A.1	Bibliography	35
A.2	Software/Tools	36
A.3	Acknowledgments	36
B	Glossary	37
C	User Guide	37

List of Figures

1	Diagram of an aquifer[8].	8
2	Example of the three common types of mathematical models for the semi-variogram with the same range/sill.	10
3	The differential of the variance of error with respect to the weights. Kriging seeks to minimize this variance to find the 'best' estimator.	11
4	'Normal' Gaussian Distribution. The mean is the interpolated value.	12
5	Computationally finding experimental semi-variogram.	13
6	Screen capture of the plot section of the window showing an optimal experimental semi-variogram, created from sample data.	13
7	Illustration of concept behind Givens Rotation. The i axis is rotated so as to make point P lie upon it, zeroing its j' coordinate.	15
8	Algorithm developed to randomly iterate through unknowns. Elements are swapped to the back part of the array as they are randomly selected from the front part. . . .	18
9	Variation on the random iteration algorithm, grouping 'failed' elements in the front to be retried after all the others have been iterated over.	19
10	Break down of Givens Rotation to be done in parallel on the GPU where rows 1 and 2 are those affected by the current rotation.	21
11	wxFormBuilder workspace	25
12	The x-z plane for the 2D grid. The y coordinates are set based upon the value at each point.	26
13	An RGB Cube - a graph of the colors with the red, green, and blue components on each axis.	27
14	A cropped screen capture of a 3D terrain generated for sample data.	28
15	An example of the heightmap rendered for the same sample data as the terrain. . . .	28
16	Two dimensional cross-section of boreholes with different elevations.	30
17	Location of the boreholes from the case study.	31
18	Semi-variogram of each of the data sets for the two boreholes. These are focused on the y direction as there is only one known point in the x direction.	32
19	Height maps produced for inverse distance weighting interpolation between boreholes. From top to bottom they are: resistivity, total porosity, SGR, and effective porosity.	33
20	Visual results of small section of borehole interpolated with kriging.	33
21	Data inserted	37
22	Semivariogram plotted, mathematical models chosen	38
23	Model type, range and sill, scales, interpolation method, number of runs selected . .	38
24	Final screen with a terrain map	39
25	Left->Height Map, Right->Terrain	39

Executive Summary

This project sought to develop a windows-based application to perform geostatistics, with a focus on its application to finding aquifers and other groundwater sources. Geostatistics is a branch of applied statistics used to calculate plausible values to fill the gaps in fragmented data sets. It depends on the idea of spatial correlation - that values located proximately are more likely to be similar. The application in hydro-geology was chosen because of New Mexico's dependency on groundwater; a case study was set up and real world data acquired from local boreholes.

This project encompassed an impressive feat of coding, incorporating C++, wxWidgets, OpenGL, and OpenCL. C++ is used for the computational and to interface with the user-interface done with wxWidgets. Visualization was done using OpenGL, and the beginning stages of optimization utilized OpenCL to run on the graphics processor.

While this project is not the first computer programming to be accomplished in the field of geostatistics, this project is a foundation for future studies. In the course of the year, several original algorithms were developed as well as integration of established methods, resulting in a working application that can reliably perform geostatistics of small problems. The case study on a real problem was partially successful and provided invaluable insights into future work.

1 Introduction

1.1 Problem Statement

A lack of complete data is a common problem faced across many fields of study. The solution is to estimate these unknowns, but making an accurate approximation becomes much more complex than simple means. The practice of this approximation is more difficult than the simplicity of the theory behind it. Geostatistics is a branch of statistics that can be used to make reliable predictions. It is based on the theory that data proximately located is more likely to be related. If the data is related then the unknowns can be approximated because the distance between data points would tell one how similar they should be.

Geostatistics is an effective method of filling in these “gaps” to logically create plausible data for the unknown points. This can be important especially in computer modeling where data is needed for every point, geostatistics can simplify the process.

The application of geostatistics to the discovery of aquifers was chosen because of New Mexico’s shortage of water. Water is a very valuable commodity in the drier and more polluted regions of the world and the easier discovery of more water sources would help many people. To find new groundwater sources in such large expanses of land by testing every mile or every half mile would be difficult and inefficient. Using geostatistics to fill in gaps in the landscape will allow geologists and hydrologists to take far fewer samples and come up with more correct results. This will lessen the time and expense of finding groundwater sources, benefiting both economic and hydrological issues.

1.2 Objective

The purpose of this project was to write a windows-based program to perform geostatistics. The program was designed to approximate unknown values and show the detail of the terrain values in both color and height. There are many possible applications for geostatistics, and thus the usefulness of the program. The focus is on using a geostatistical model to find aquifers without taking inordinately large numbers of samples for a given area. To accomplish this goal a profusion of code had to be written for the many different facets of the program. The initial code to estimate the data using geostatistics was written in C++, the user interface was generated in wxFormbuilder (creating wxWidgets C++ code), and the graphics were rendered with OpenGL. The team wished to create a working program utilizing all three different programming languages that would accurately predict unknown values for a data set. These unknowns pertain to aquifer data so as to find more groundwater sources and alleviate problems in New Mexico and the rest of the world.

The program should reduce the time and money spent on geological surveying by a sizable margin and can be changed minimally to be applied to other problems.

1.3 Background

1.3.1 Geostatistics

Geostatistics is a branch of statistics used to predict unknown values at specific locations, using the concept of spatially correlated data. That is, two values physically near each other are more similar

than two values farther apart. For example, in soil composition, samples taken closer together are more likely to be made up of similar minerals.

Geostatistics, originating in mining for the discovery of precious stones and metals, was first recognized as a reputable field theory in the 1960s in the French work “Theory of Regionalized Variables” which paved the way for inspirational work in the new discipline. Many changes were made to the math used in geostatistics and eventually it became applicable to many different fields besides mining. Now such employments as picture reconstruction and epidemiology are utilizing geostatistics[2].

To understand how geostatistics works one must understand the theories at the heart of the process. The Theory of Regionalized Variables states that it is possible to make a model of the spatial structure from known data and then use those known values to estimate the unknown ones[6]. The unknowns can be estimated because of the theory that data is spatially correlated. These are the underlying precepts behind geostatistics; the theories that make all others possible.

These postulates are used to determine the value of a given property in specific materials. This is done by applying the Theory of Regionalized Variables. There are two parts to regionalized variables:

- a *random* aspect, the unpredictable variation from point to point
- a *structured* aspect, the prevalent regional trend

The random aspect is the deviation from the normal that will throw off an approximation whereas the structured aspect is the normal trend which allows for the use of geostatistics in the estimations of unknown values.

Like most fields, there is some specialized language used in geostatistics. These terms will be defined as they appear, but are also described in the glossary (Appendix B on page 37)

1.3.2 Aquifers

The chosen application was locating aquifers. Background information was needed to discover what characteristics to search for as an indication of an aquifer.

An aquifer is an underground layer of water-bearing permeable rock (Figure 1), which can be tapped by a well. The above diagram shows how the location of the water table is relative to the surface and to the surrounding geological points. An aquifer is a valuable commodity as a water source because, by definition, it readily transmits water to wells and springs. This means that it will not be stagnant and undrinkable. Also because of the location of aquifers underground, the water cannot evaporate before its use. Unfortunately, aquifers are difficult to locate and can be contaminated. Aquifers are more likely to be closer to the surface because of the porous and permeable rocks there. Porosity refers to a rock’s ability to retain water, while permeability is the capability of a porous rock to permit the flow of fluids through it[12]. The permeability and porosity generally decrease for larger distances from the surface since the cracks and fissures in a become diminished and close up as a result of the pressure of the rock overhead. However, this is not always the case and usable aquifers have been found in all surface depths[8].

More valuable results can be garnered from porosity and permeability because they are more reliable variables to use in a geostatistics model. This is because porosity and permeability more extensively affect the rock’s ability to be a potable resource[8].

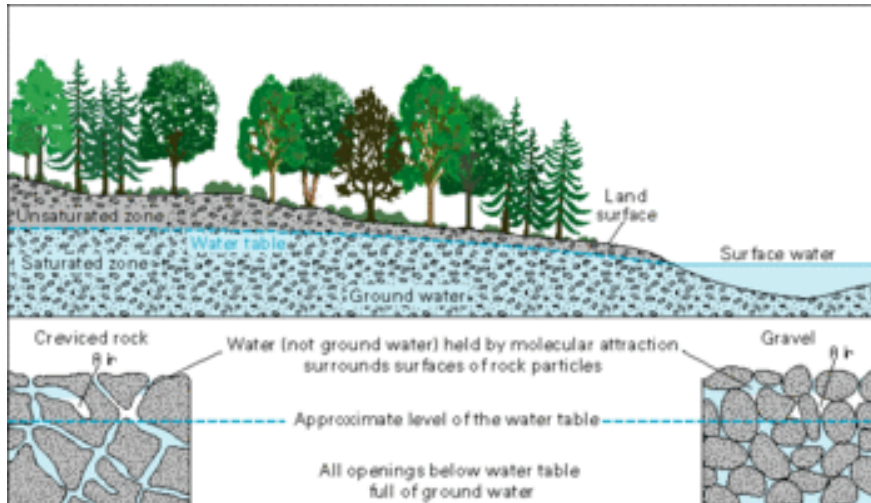


Figure 1: Diagram of an aquifer[8].

An aquifer can become contaminated as a result of human interference. Every time the water in an aquifer is used in a well, the level of the water table goes down and the water will be replaced with precipitation, known as recharge. The area in which an aquifer can benefit from the recharge is called a recharge zone. The larger an aquifer's recharge zone, the more wells can be drilled from it and the more often people can pump water for them. However, the larger an aquifer's recharge zone is the more opportunity for the aquifer to be contaminated. If an aquifer is contaminated it cannot be used thus creating a water shortage problem for its patrons.

Humanity's interference with the water table can also result in an aquifer's water pressure decreasing at an alarming rate[8]. If one were to pump too much water out of a well without allowing it its recharge period then surrounding wells could also go dry. This creates a problem for rural communities that depend on groundwater, making aquifers very sought-after resources[5]. Aquifers are a solution to the dire problem of acquiring water in the drier, more rural regions of the world. The case study of this project seeks to make them available by predicting their locations.

2 Mathematical Models

Geostatistics can be outlined with two main goals: to identify the spatial properties of the variable and to estimate gaps in incomplete data from the surrounding samples. These purposes are related, as characteristics of the spatial structure can be used to estimate unknowns. This is done by 1) constructing a semi-variogram, and 2) interpolating through the use of either inverse distance weighting or kriging.

2.1 Semi-variogram

The idea of spatial correlation discussed in the introduction to geostatistics is fairly intuitive. It makes sense that a value close to the unknown will be more similar to it than a value farther away. The semi-variogram is a way to quantify the variance in the values over space. It is fundamental to the idea of spatial correlation, and a crucial part of geostatistics.

The semi-variogram is unique for each material. Looking at many pairs of data at points about the same distance apart can provide an expected difference in value for a given distance. Described mathematically it is[2]:

$$\gamma^*(h) = \frac{\sum [y(x) - y(x+h)]^2}{2n}$$

where $\gamma(h)$ is the semi-variogram¹ as a function of distance h between the data points, $y(x)$ and $y(x+h)$ are the values at locations x , and x plus the distance h , and n is the number of pairs of samples with distance h separating them.

These points are plotted with distance on the horizontal axis and semi-variogram on the vertical axis.

There are several specific terms associated with the semi-variogram. The distance at which the graph plateaus is called the range of influence, or simply the range. Any points farther apart than the range are completely uncorrelated, and thus are not helpful in accurately interpolating a value. The semi-variogram at that point is referred to as the sill. In experimental semi-variograms, it is possible that there will be a discontinuity at the origin called a nugget. In theory, this should not happen because the value at a point is equal to its own value; however, measurement errors and a random influence between the points can cause a nugget.

One of the things a semi-variogram can reveal about the data it represents is its isotropy. The material can be isotropic, meaning the spatial correlation is equal independent of the direction. If direction is an influence, it is anisotropic. Wood is a great example of this. There is a greater range so the relation extends much farther along the grain than against it.

There are several types of mathematical models which can be matched to the semi-variogram obtained from the data. A simple semi-variogram can be represented by a single type, but they can be combined for more complexity. Three of the most commonly used mathematical models are[2]:

Spherical

$$\gamma(h) = \begin{cases} C \left(\frac{3h}{2a} - \frac{1}{2} \frac{h^3}{a^3} \right) & h \leq a \\ C & h > a \end{cases}$$

Exponential

$$\gamma(h) = C \left[1 - \exp \left(-\frac{h}{a} \right) \right]$$

Gaussian

$$\gamma(h) = C \left[1 - \exp \left(-\frac{h^2}{a^2} \right) \right]$$

2.2 Interpolation

There are several different approaches to interpolating. Generally, this is done with the general equation for a weighted average shown below:

¹The distinction between the *variogram*, $2\gamma(h)$, as opposed to the semi-variogram is important, and not always clear if semi-variograms are inattentively called just variograms.

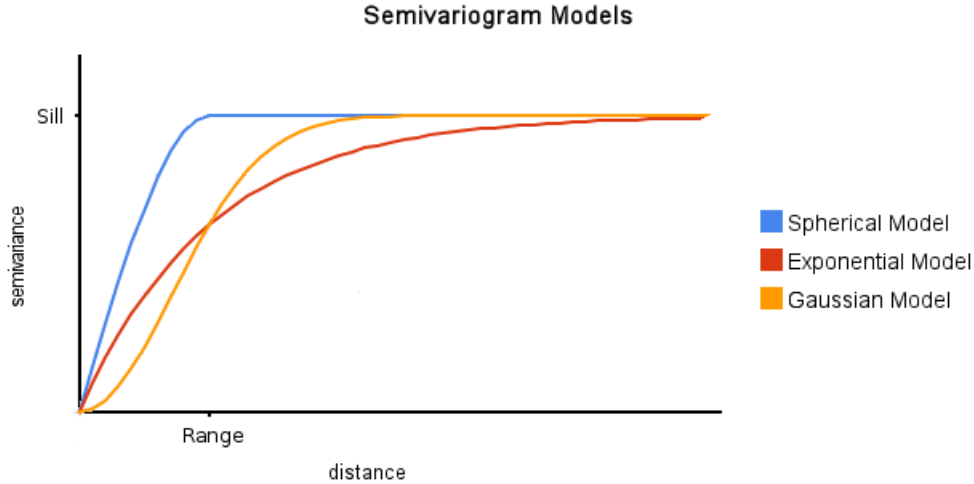


Figure 2: Example of the three common types of mathematical models for the semi-variogram with the same range/sill.

$$F(x, y) = \sum_{i=1}^n w_i f_i$$

which basically says that the value of the unknown point is a summation of the values of the points it is being interpolated from times its weight. The differences occur in determining the weights.

2.2.1 Inverse Distance Weighting

One of the simplest methods is Inverse Distance Weighting (IDW). The weights are purely dependent on the distance. The inverse of the distance for each of the points within range is found. If a point is 4 units from the unknown, it would be $\frac{1}{4}$. Then they must be summed and scaled to one. So the weight of each point is:

$$w_i = \frac{d_i^{-1}}{\sum_{j=1}^n d_j^{-1}}$$

where d is the distance for each point i of n points. The distances can be scaled to provide larger weights in some directions than others as determined in the semi-variogram this is discussed in more detail in Section 3.3. This method works fairly well for its simple approach and is a good comparison method to the kriging.

2.2.2 Kriging

Kriging is an interpolation method unique to geostatistics. It works by finding the “best” estimate. An explanation of how this is done will be given in terms of an unknown value T at a point A , as adapted from *Practical Geostatistics*[2].

If the value at the closest point is used as an estimation of T , it will incur an estimation error ε which is a measure of the difference between T and the estimated value T^* :

$$\varepsilon = |T - T^*|$$

Assuming there is no local trend, as the number of estimations increases towards infinity, the average error will approach zero. So, theoretically:

$$\bar{\varepsilon} = 0$$

The reliability of an estimator is rated by the spread of the errors. A 'good' estimator has errors consistently close to zero. If they range more widely, than the estimator is unreliable. The spread can be measured by the standard deviation of the estimation error - the standard error.

Consider the definition of standard deviation σ , the square root of the variance.

$\sigma^2 = \frac{\sum (X - \mu)^2}{N}$, the average of the squares of the difference from the mean. In case of the variance of errors, it follows that:

$$\begin{aligned} &= \text{average of } (\varepsilon - \bar{\varepsilon})^2 \\ &= \text{average of } \varepsilon^2, \text{ since } \bar{\varepsilon} = 0 \\ &= \text{average of } (T - T^*)^2 \end{aligned}$$

It is impossible to calculate these values directly since the actual value is unknown. A closer look at the definition provides a solution. The value of the point closest to point A is used for the estimator, and should vary from the actual value dependent on the distance from A . This expected difference is described by the variogram exactly, the average of the squared differences. Thus, the mathematical semi-variogram chosen to represent the spatial structure can be used to estimate the difference (multiplying by two yielding the variogram). So, finally, the variance of the errors is:

$$\sigma_\varepsilon^2 = 2\gamma(h)$$

As the estimate grows more complex with the addition of other points, the variance of errors is given as a weighted average of the variogram of each of them.

$$\sigma_\varepsilon^2 = 2 \sum_{i=1}^n w_i \bar{\gamma}(P_i, A)$$

$\bar{\gamma}(P_i, A)$ is the average semi-variogram, as defined by the mathematical curve, between interpolation point P and the point being estimated, A . Kriging is unique in that it directly seeks to find the 'best' estimate - that having the smallest estimation variance. The only values free to be altered are the weights of the weighted average, so the estimation variance is being minimized with respect to the weights. A minimum can be found by setting the differential equal to zero (i.e. the slope is zero as in Figure 3):

$$\frac{\partial \sigma_\varepsilon^2}{\partial w_i} = 0 \quad i = 1, 2, 3, 4 \dots n$$

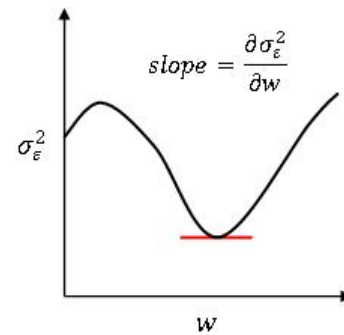


Figure 3: The differential of the variance of error with respect to the weights. Kriging seeks to minimize this variance to find the 'best' estimator.

While this will provide weights for the desired minimum, the sum of the weights must also be explicitly set to one so the related estimator will be a whole.

$$\sum w_i = 1$$

This added constraint consequently over-determines the system of equations - n weights, or variables, and $n + 1$ equations. A Lagrangian Multiplier is introduced to balance this out. Rather than simply the estimation variance to be minimized, it is the term:

$$\sigma_\varepsilon^2 - \lambda \left(\sum w_i - 1 \right)$$

When the sum of the weights is equal to one, $\sum w_i - 1 = 0$, thus nullifying the λ and taking the smallest variance as defined. The expanded system equations for three points is defined as:

$$w_1\bar{\gamma}(P_1, P_1) + w_2\bar{\gamma}(P_1, P_2) + w_3\bar{\gamma}(P_1, P_3) + \lambda = \bar{\gamma}(P_1, A)$$

$$w_1\bar{\gamma}(P_2, P_1) + w_2\bar{\gamma}(P_2, P_2) + w_3\bar{\gamma}(P_2, P_3) + \lambda = \bar{\gamma}(P_2, A)$$

$$w_1\bar{\gamma}(P_3, P_1) + w_2\bar{\gamma}(P_3, P_2) + w_3\bar{\gamma}(P_3, P_3) + \lambda = \bar{\gamma}(P_3, A)$$

$$w_1 + w_2 + w_3 = 1$$

On the left side of the equation are the weights times the variance between each point with each other point, and on the left is the variance between said point and the unknown. The system of equations follows the same pattern for any number of points used to interpolate. Solving these equations is a computational problem addressed in Section 3.2.

2.3 Sampling from Gaussian Distribution

If the goal of the interpolation is not to be accurate but to generate a sample set of data to be used, than it may be desired to address the random aspect of a regionalized variable. To give the impression of a degree of random variance in each point, a Gaussian Distribution is randomly sampled.

The interpolation gives the mean of the distribution, or the most likely value, but the value is free to vary from this. The magnitude of this variance is based on the variance of the unknown to the closest point as determined from the semi-variogram. This is more of a stylistic choice, and results in unknowns interpolating from farther afield points having a larger random element. This sampling is set up as the mean (μ) - or most likely value - as being the

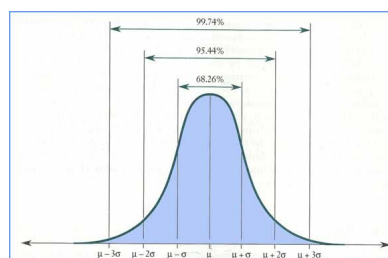


Figure 4: 'Normal' Gaussian Distribution. The mean is the interpolated value.

interpolated estimator. The standard deviation (σ) is the square root of the variance found from the closest point. Values closer to the mean are more likely to be sample because of the bell shape of the curve, and the smaller the variance the smaller the range of possible values.

3 Computational Model

Since our program is windows based, the computational method is much more segmented. Each part is driven by user generated events (e.g., clicking buttons, typing).

After the data is input, there is a vector array of the grid points whose values are unknown. The user can interact with a plot to create a semi-variogram from the known data.

3.1 Semi-variogram

The program must calculate both experimental and mathematical semi-variograms. The experimental semi-variogram is somewhat simplified because the data is already broken up into equally spaced points. It did not take long to develop the basic method that was used. For convenience, the equation will be repeated here.

$$\gamma(h) = \frac{\sum [y(x) - y(x+h)]^2}{2n}$$

Part of the process is actually simplified because the data is already divided into equally spaced points. Each row or column begins at the first element and “jumps” over h points. If the values at both points are known, the semi-variogram is calculated.

This process is continued by incrementing up the row by one until there are no longer enough elements to skip h . Then h is increased.

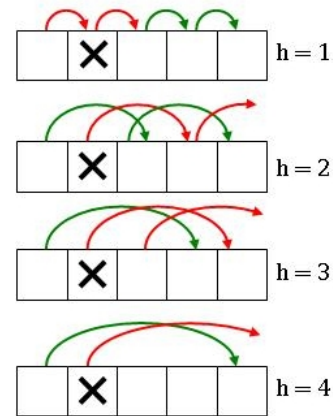


Figure 5: Computationally finding experimental semi-variogram.

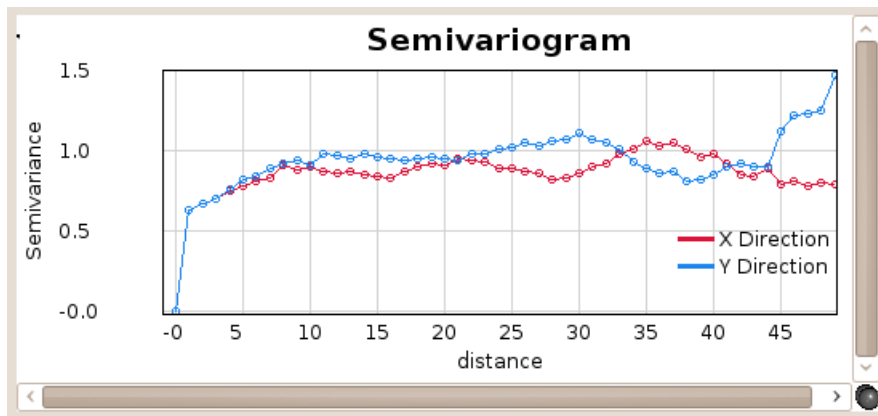


Figure 6: Screen capture of the plot section of the window showing an optimal experimental semi-variogram, created from sample data.

3.2 Solution of Kriging Equations

The kriging equations involved a more complex solution than IDW, using matrices to solve for all of the unknowns in the system. The added constraint on the sums makes the system of equations overdetermined, meaning there are more equations than variables. QR decompositions are a common solution for least squares problems with over-determined systems of equations[3].

QR factorization of the coefficient array was used to solve for the weights, and was computed using Givens rotations. QR factorization of a square matrix $A \in \mathbb{R}^{n \times n^2}$ is given by $A = QR$, where Q is orthogonal and R is upper triangular. The definitions of these special types of matrices are as follows:

A square matrix $Q \in \mathbb{R}^{n \times n}$ is orthogonal if $Q^T Q = Q Q^T = I_n$, meaning its inverse is also its transpose³. An upper triangular matrix, also called right triangular, is also square ($n \times n$), with all the entries below the main diagonal zero:

$$U = \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\ 0 & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\ 0 & 0 & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \ddots & u_{n-1,n} \\ 0 & 0 & 0 & 0 & u_{n,n} \end{bmatrix}$$

Givens rotations can be used for selectively zero elements, and calculate the decomposition of a matrix into its Q and R factors. Multiplication by a rotation matrix⁴ performs a rotation in Euclidean space. To visualize a matrix geometrically, consider each column to be a set of coordinates to define the location of a point. Multiple points create a set of columns, a matrix, with each row the coordinates in the same dimension. So the matrix $\begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}^T$ performs a counterclockwise rotation of an angle θ about the origin of an x-y-plane - or alternatively viewed as the rotation of the coordinate system axes in the opposite direction. A rotation can be performed on a larger scale by expanding the previous rotation matrix to:

$$G(i, j, \theta) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & \cos \theta & \cdots & \sin \theta & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -\sin \theta & \cdots & \cos \theta & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}$$

which is an identity matrix with the following substitutions: $g_{ii} = \cos \theta$, $g_{ij} = \sin \theta$, $g_{ji} = -\sin \theta$, $g_{jj} = \cos \theta$. The only rows affected are i and j , the others will remain the same, and thus may be

²This denotes the vector space of all real n -by- n matrices, essentially saying any matrix with the given dimensions.

³The transpose of a matrix is denoted with a superscript T.

⁴It should be noted that rotation matrices are orthogonal and have a determinant of one.

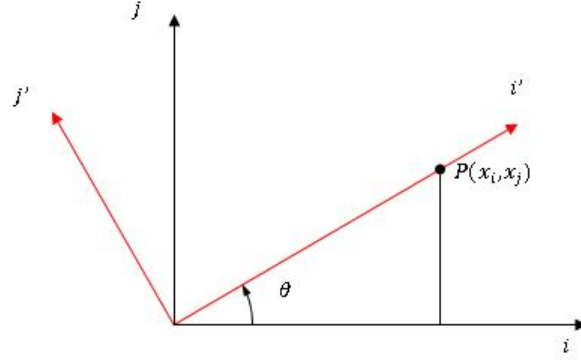


Figure 7: Illustration of concept behind Givens Rotation. The i axis is rotated so as to make point P lie upon it, zeroing its j' coordinate.

ignored.

A Givens rotation sets the angle to rotate the axis so the selected point lies on it, zeroing out the value of the other coordinate. If $x \in \mathbb{R}^n$, and $y = G(i, j, \theta)^T x$, then, by matrix multiplication:

$$y_k = \begin{cases} x_i \cos \theta - x_j \sin \theta & k = i \\ x_i \sin \theta + x_j \cos \theta & k = j \\ x_k & k \neq i, j \end{cases}$$

y_j can be forced to be zero when it lies on the perpendicular axis. Figure 7 illustrates the right triangle created by the previous axis and the new one that should run through point $P(x_i, x_j)$. This creates the desired angle of rotation. By directly using the definitions of the trigonometric functions used in the rotation, calculation of θ can be bypassed entirely. The Pythagorean Theorem ($c^2 = a^2 + b^2$) gives the length of the hypotenuse, and the sides of the triangle are known from the coordinates.

$$\cos \theta = \frac{adj}{hyp} = \frac{x_i}{\sqrt{x_i^2 + x_j^2}}$$

$$\sin \theta = \frac{opp}{hyp} = \frac{-x_j}{\sqrt{x_i^2 + x_j^2}}$$

Substitute these definitions into the expression for y_j :

$$x_i \sin \theta + x_j \cos \theta$$

$$\frac{-x_j}{\sqrt{x_i^2 + x_j^2}} x_i + \frac{x_i}{\sqrt{x_i^2 + x_j^2}} x_j$$

$$\frac{-x_j x_i + x_i x_j}{\sqrt{x_i^2 + x_j^2}} = 0$$

And they result in zero because the first term of the numerator has a negative, making them additive inverses. This only happens in the specific case that was intentionally set up.

These Givens rotations can be used to find the QR factorization used to solve a matrix equation $Ax = b$. Each element of the coefficient matrix A where i is greater than j would be zeroed, creating an upper triangular matrix.

From the definitions of sine and cosine and the Pythagorean Theorem, they can be found as:

$$\cos \theta = \frac{j}{h} = \frac{x_i}{\sqrt{x_i^2 + x_j^2}} \quad \text{and} \quad \sin \theta = \frac{i}{h} = \frac{-x_k}{\sqrt{x_i^2 + x_j^2}}$$

Now the matrix equation has been put into the form $Ux = b$, it can be solved using back substitution. For a 2-by-2 example:

$$\begin{bmatrix} u_{1,1} & u_{1,2} \\ 0 & u_{2,2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

The bottom row has only one unknown, since the other variable is zeroed, and can be solved directly. Once that value is known, it can be substituted up into the next row so there is again only one unknown which can then be found. Starting at the bottom row, the unknown x 's can be solved for sequentially:

$$u_{2,2}x_2 = b_2 \text{ is solved algebraically to be } x_2 = b_2/u_{2,2}$$

$$u_{1,1}x_1 + u_{1,2}x_2 = b_1 \text{ is } x_1 = (b_1 - u_{1,2}x_2)/u_{1,1}$$

This back substitution can be represented by[3] :

$$x_i = \frac{b_i - \sum_{j=i+1}^n u_{i,j}x_j}{u_{i,i}}$$

This process provides the values of the weights in the weighted average. The corresponding code for a coefficient matrix A, right hand vector B, and matrix size n, as based off of the psuedo code in *Matrix Computations*[3] is given in Listing 1. The solution is stored in vector x.

Listing 1: Matrix Solver using Givens Rotations

```
void GeoData::QRGivens(double **A, double *B, int n, double *x) {
    double c, s, tau, tau2;
    int i, j, k;
    for(j = 0; j < n; j++) {
        for(i = n-1; i > j; i--) { //loop to zero all elements in lower triangle
            /* Determine variables for rotation */
            if(A[i][j] == 0.0) { //already 0 - don't change
                c = 1.0;
                s = 0.0;
            }
            else if(abs(A[i][j]) > abs(A[j][j])) {
                tau = -A[j][j]/A[i][j];
                s = 1.0/sqrt(1.0+SQ(tau));
                c = s*tau;
            }
            else {
                tau = -A[i][j]/A[j][j];
                c = 1.0/sqrt(1.0+SQ(tau));
                s = c*tau;
            }
        }
    }
}
```

```

    for(k = 0; k < n; k++) { //perform rotation on elements in two
        affected rows
        tau = A[j][k];
        tau2 = A[i][k];
        A[j][k] = c*tau - s*tau2;
        A[i][k] = s*tau + c*tau2;
    }
    /* Same rotation done on right hand solution vector */
    tau = B[j];
    tau2 = B[i];
    B[j] = c*tau - s*tau2;
    B[i] = s*tau + c*tau2;
}
}
/* Back substitution stores solution in x */
for(j = n-1; j >= 0; j--) {
    x[j] = B[j];
    for(i = n-1; i > j; i--) {
        x[j] -= x[i]*A[j][i];
    }
    x[j] /= A[j][j];
}
}
}

```

3.3 Anisotropy

Anisotropy is the property of being directionally dependent (as opposed to being isotropic). In geostatistics, this means having different spatial correlation in different directions. Wood is a good example of this characteristic. It is evident, even to the human eye, that it has a higher degree of correlation along the grain than against it. This might not be so apparent in the various qualities of different substances. Therefore, anisotropy must be identified using the semi-variogram. The plots of the experimental semi-variograms for the two directions will have different ranges of influence.

The solution is to make the ranges appear to be the same. This means adjusting the measurements so that one 'unit' in the semi-variogram may be 5m horizontally and 25m vertically.

The calculations rely on a distance method to calculate how far one point is from another. This is simple since they use a coordinate system. The distance formula is:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

If the formula is modified, it can account for the scaling of measurement. If s is the scale factor, in the form of a decimal percent (i.e. to have half the range, s would be 0.5), then the new formula can be expressed as:

$$d = \sqrt{[s_x(x_2 - x_1)]^2 + [s_y(y_2 - y_1)]^2}$$

The magnitude of the distance in each direction is scaled before the rest of the distance formula is performed. This scaling effect has also been applied to relative distances in the two directions. If it is one foot from one cell to the next horizontally, but two feet vertically, then the ratio is included

in the scale to make it a default of isotropic - the point above the unknown really is twice as far away as the one to the right of it.

3.4 Sampling from Gaussian Distribution

The Gaussian Distribution was fairly simple to set up with the use of a method from *Numerical Recipes in C*[9]. Once the interpolated value is set, the nearest point is found by searching through the list of points in range and the variance computed for the distance. The random sample returned from the method can be applied to the specific case.

```
double a, b;
a = estimate.getValue();
b = GetVariogram(estimate, GetClosestPoint(estimate));
return a + gaussRandom()*b;
```

The method `gaussRandom()` returns a sample from a Gaussian Distribution with a zero mean and unit variance. Multiplication by b stretches the curve horizontally and adding a shifts it horizontally.

Since this sampling was not important to the focus of this project, it has not been tied completely into the user interface: there is not an option in the window for it. The code needed to compute it is completed, but the method to apply it to each of the unknowns by choice of a user was not updated with the other code changes.

3.5 Multiple Points and Runs

Thus far, the discussion has been limited to the interpolation of a single point. To find all of the unknowns, they are ordered randomly to be calculated. This sequence is important because the points are interdependent - once an estimated value is found it is used in the interpolation of others.

Because of the random element of order, the answers will vary between runs, making it necessary to run multiple times. This repetition and integration of results, in the form of a mean, has been automated.

An algorithm had to be developed to randomly iterate through an array of the unknown points. Each element is a structure which contains the index of the unknown and a sum of the answers - which can then be divided by the number of runs to find the mean.

This is all done in a single array. A random index between zero and the maximum size of the array is chosen. Once the interpolation has taken place and added to the sum, the element is switched with the last element. The next random index is chosen, but this time excluding the final element which has already been found - that is between zero and one less

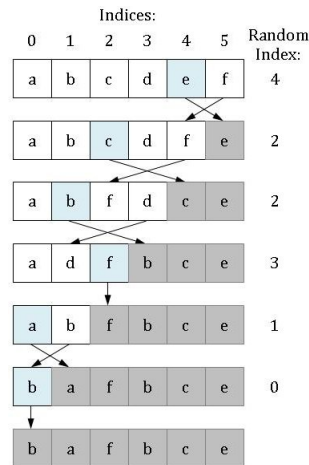


Figure 8: Algorithm developed to randomly iterate through unknowns. Elements are swapped to the back part of the array as they are randomly selected from the front part.

than the array size. This next element is switched with the second to last element, creating a segment at the end of the array of the elements that have already been used. The next random element is searched for between zero and the maximum minus the number of completed elements.

This algorithm works fine until the problem of failures to calculate the points arises. If there are no points within the range of the current unknown, it has nothing to interpolate from. It must be skipped until later when more points have been calculated.

3.6 Optimization

Large scale problems are the norm in practical computing, calling for faster calculations to curb the lengthy run-times. Due to the sectionalized nature of the code, these beginning stages of optimization have been done within the separate methods. Later work may attempt to make this more streamlined in order to further improve the speed.

3.6.1 Variation on Random Iteration Algorithm

Though it is not necessarily characteristic of large problems, sparse data can become computationally expensive in the random iteration through the unknown points. In the original method outlined on Section 3.5, if a point fails - that is there are no points in range from which to interpolate - the unknown is left where it is in the array and the count of finished elements does not increase. This works when there is ample data and these types of points are rare. Another point is picked and eventually it will be successfully calculated as the points around it are found.

In a set of sparse data, however, there are so many points with nothing in range that the random index could continually hit these points and no progress would be made. An alternative algorithm was created, which, while it does not ensure equal chances in determining the random path, prevents already failed elements from being selected again before more points are filled in. This prevents repetitive sampling which may waste computation time.

This was accomplished by setting off elements at the beginning of the array, similar to the one on the end. If the calculation of the unknown at a random index is successful - that is there were points in range to interpolate from - it is swapped into the back section of the array. If it fails,

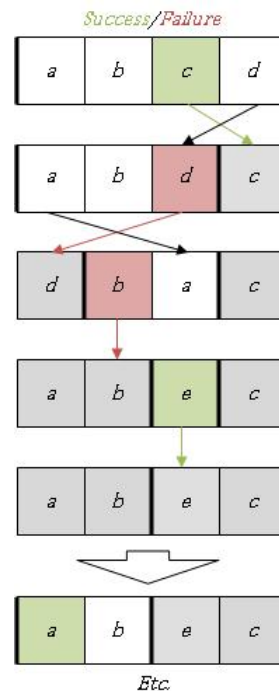


Figure 9: Variation on the random iteration algorithm, grouping 'failed' elements in the front to be retried after all the others have been iterated over.

it is swapped into the front. Random indices are always selected from between the two boundaries of the separated sections by finding a random number for the range and offsetting it from the front:

```
index = rand() % (size-i-front) + front;
```

Once the boundaries converge, leaving no unknowns in the middle which have not been iterated over, the front boundary is reset to the beginning of the array. This puts all the previously failed elements in the middle to be retried. This continues until all the elements have been successfully completed and moved to the back - or if the boundaries converge with no new successfully completed unknowns, indicating the elements that are left are impossible to calculate with the current parameters.

The variation on the algorithm is implemented in the following fashion:

Listing 2: General Use Implementation of Variation of Random Iteration Algorithm

```

success = 0;
front = 0;
size = array.size();

for(i = 0; i < size; i++) { //iterator to determine when all elements are completed
    if(front == size - i) { //check if the boundaries have converged
        if(success < 1) //if there have been no successes here, the rest are impossible
            send error
        front = 0 //reset boundary and success count
        success = 0
    }
    index = rand() % (size-i-front) + front //in C++, find random index between
        boundaries
    if(calculation returns successful) {
        swap array[index] with array[size-i-1]
        success++
    }
    else { //if the calculation failed
        swap array[index] with array[front]
        i-- //compensate for automatic increment of i, it wasn't successful
        front++ //move up boundary of front section
    }
}
}

```

Since there are potential consequences in the order of the random path, this alternative is only used with the selection of a sparse data option in the application window.

3.6.2 Transferring Matrix Solver to the GPU

The Givens Rotation QR decomposition was parallelized using OpenCL to send it to the Graphic Processor Unit (GPU). Large problems continued to use inordinate amounts of time without reaching completion. A smaller sized test problem was run to identify which methods were most computationally expensive. A significant 55.56% of the total time was spent in the QRGivens method (the solver for the kriging equations), which was also much larger than the next largest at 7.41%. This made the matrix solver a clear target for optimization.

The speed-up gained by operating on a GPU is mainly due to the parallel computation - the same process is done on multiple data elements simultaneously[7]. The speed of calculations on the GPU make loading data the primary concern. The coefficient and right hand vector of the matrix equation are passed in globally. For every rotation, only two rows are affected, as discussed in the description of Givens Rotations (Section 3.2) . Thus the rows can be loaded into local memory by

pairs, providing faster access.

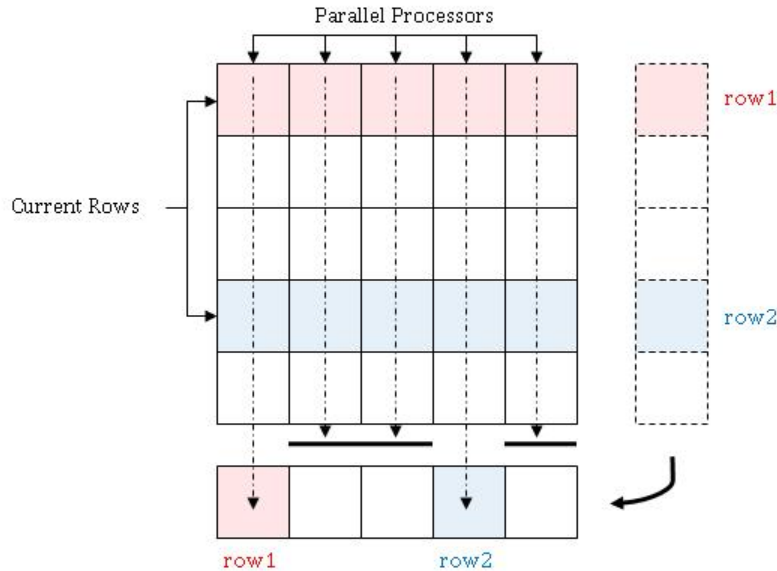


Figure 10: Break down of Givens Rotation to be done in parallel on the GPU where rows 1 and 2 are those affected by the current rotation.

Parallelization can only be used where the values are independent of each other. Once the coefficients of the rotation are found, they can be applied to each of the elements individually - that is in parallel. This break down is shown in Figure 10. The current kernel implementation is shown in Listing 3.

Listing 3: OpenCL Givens Rotation Kernel

```

#define A(j,i) Coeff[j*npadded+i].s0
#define SQ(a) ((a)*(a))
__kernel void QRGivensGPU_kern(
    const int npadded,
    const int n,
    __global float2* BX,
    __global float2* Coeff,
    __local float* row1,
    __local float* row2,
    __local float* B)
{
    int giX = get_global_id(0);
    int tiX = get_local_id(0);
    int ngX = get_global_size(0);
    int ntX = get_local_size(0);
    float c, s, tau, tau2;
    int i, j, k;
    if(giX < n) { //Only for processors that were not added for padding
        B[giX] = BX[giX].s0;
        for(j = 0; j < n; j++) {
            row1[giX] = A(j,giX); //Each processor sets value in row1 from
                corresponding A
            for(i = n-1; i > j; i--) {

```

```

row2[giX] = A(i,giX);
barrier(CLK_GLOBAL_MEM_FENCE); //Force wait until all processors
are done
/* Set rotation values */
if(row2[j] == 0.0f) {
    c = 1.0f;
    s = 0.0f;
}
else if(fabs(row2[j]) > fabs(row1[j])) {
    tau = -row1[j]/row2[j];
    s = 1.0f/sqrt(1.0f+SQ(tau));
    c = s*tau;
}
else {
    tau = -row2[j]/row1[j];
    c = 1.0f/sqrt(1.0f+SQ(tau));
    s = c*tau;
}
/* Perform rotation on each element - parallel */
tau = row1[giX];
tau2 = row2[giX];
row1[giX] = c*tau - s*tau2;
row2[giX] = s*tau + c*tau2;
/* Perform rotation on appropriate elements of solution vector */
if (giX == i || giX == j) {
    tau = B[j];
    tau2 = B[i];
}
barrier(CLK_GLOBAL_MEM_FENCE);
if (giX==j) B[j] = c*tau - s*tau2;
if (giX==i) B[i] = s*tau + c*tau2;

A(i,giX) = row2[giX]; //Put updated values to the coefficient array
}
A(j,giX) = row1[giX];
}
/* Back substitute, putting answer into second vector component of BX */
barrier(CLK_GLOBAL_MEM_FENCE);
for(j = n-1; j>=0; j--) {
    for(i = n-1; i > j; i--) {
        B[j] -= B[i]*A(j,i);
    }
    B[j] /= A(j,j);
}
BX[giX].s1 = B[giX];
}
}

```

4 Code

4.1 Overview and Structure

This is a large project and, especially with the GUI, there is a significant amount of code with a complex structure. There are several key parts that will be discussed in more detail in the next sections. This project was written primarily in C++, but also incorporated several different packages: wxWidgets (and its add-ons), OpenGL, and OpenCL.

The wxWidgets code which creates the windowing is generated using wxFormBuilder, but it is not used directly. In fact, the programmer should not hand edit it at all. Instead, a child class is created, inheriting the frame design and objects within in. It is in this class that the methods called on each event are implemented. The role of each class will be clarified with a description of their place in the structure and their methods.

AdamsAleAppGui is the file generated by wxFormBuilder, and actually contains several classes for the frame and each of the dialogs. Each one creates the window with the layout as designed in wxFormBuilder, but nothing is functional.

AdamsAleApp is the “main” class in the application. It calls the constructor of the frame displays it, and sets up the continuous rendering. Though these are its only tasks, they are important because external code (from the windowing) is needed to initialize the application.

AdamsAleAppFrame is the class inherited from the frame produced by the generated code. While the parent class has the layout, it is this class’s job to fill in designated spaces such as the GLCanvasPane. It makes the original design functional by animating the controls, that is, defining what should be done for different user inputs. The frame has access to all of the objects within it, so it can retrieve data from inputs, display values, and call the methods of more complex objects (GLCanvasPane, PlotCtrlPane). The “command events” generated by clicking buttons, selecting menu items, etc., are directed to here. In wxFormBuilder, corresponding methods for each event can be set. These methods are defined here, generally calling on more specific methods in other classes. Otherwise, it calls dialogs and has all the file I/O.

GeoData was developed later to hold all of the variables and methods on the computational side of the application. It is purely computational, with no references to any of the user interface. When a new model is created or a file opened, an instance of the class is created and values passed in. The frame can then call any of the methods: from the semi-variogram to interpolating points.

GLCanvasPane is the pane in the window reserved for the visualization. An instance of this class is created as part of the constructor of the frame. The pointer to the data class is passed into this class after the unknowns have been interpolated so it has access to the values of each point.

PlotCtrlPane comes from the wxWidgetsAddition wxPlotCtrl with additional methods for its specific use in plotting semi-variograms.

The sizes of these classes may be estimated by the number of lines in each of their files⁵:

File	Line Count
AdamsAleApp	93
AdamsAleAppFrame	432
AdamsAleAppGui	763
DataPoint	65
GeoData	460
GLCanvasPane	301
PlotCtrlPane	184
Total	2311

There is an immense amount of code, and it cannot be recorded here in its entirety; major sections are included in pertinent sections. It is all online in the repository used during its development and can be viewed at: <http://code.google.com/p/adams-ale/source/browse/#svn/trunk/AdamsAle>.

4.2 WxWidget Windowing

The windowing and user interaction were a significant part of the program. Since there are so many different options in geostatistics, this format allows a user to choose the method that best fits the specific problem. The majority of the wxWidget C++ code was generated with wxFormBuilder, saving the time needed to write out the simple code by hand. The rest of the code has been integrated into this main application window as functions.

Figure 11 is a screen capture of the workspace in which the user interface is created. On the left is the hierarchy in which the elements of the page can be easily arranged without disturbing the rest of the window. In the middle is a preview of the window the programmer is creating. On the far right is the “properties and events” window. The “properties” tab allows the programmer to set the size, labels, and properties of the object that was just created, while the “events” tab lets the programmer set the methods for the object. For example, the programmer can write “OnClick” enable that object to be used when it is clicked on by a mouse by adding matching code later on. This object can now be used because it has a function. WxWidgets is used for creating the windowing in which a the program can operate. This tool allows the programmer to create a window with relative ease. It creates most of the ‘cosmetic’ code, while it only requires the programmer to write the code to animate the controls. This eliminates a lot of lines a programmer must write by hand. wxWidgets is also very useful for programmers using more than one platform as it is virtually the same on a PC and Mac - though this has not been attempted with this project . Tabbed windows allow more information to be seen on the same page. Using wxWidgets along with wxFormBuilder was a good choice because of the relative ease. because of the relative ease of using it. In wxFormBuilder, the programmer starts with a frame, which is a basic window. Then a sizer is added. Sizers organize the window and lay out the graphs and similar objects. After a sizer there are virtually endless possibilities from which the programmer can choose: toolbars, graphs, data tables, and graphics panes to be filled by other programmers.

⁵Line counts are from a single point in time and will vary a little as changes are made. Header files are included in the count.

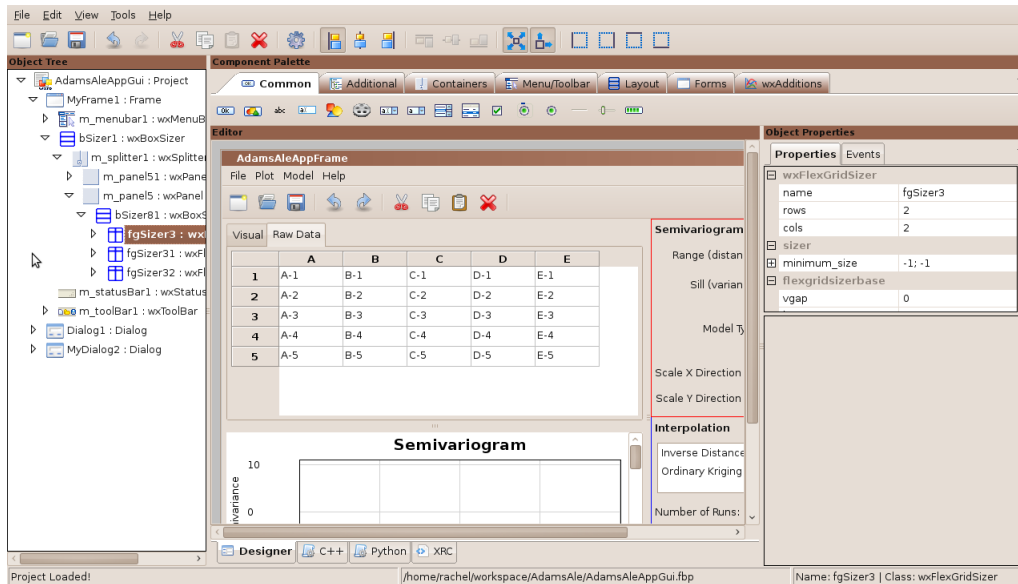


Figure 11: wxFormBuilder workspace

There are four different types of sizers: box, staticboxes, grids, and flexgrids. BoxSizers are the most common and ideal choice for a basic window. They space out the objects that are added to it equally, placing them either vertically or horizontally. A box sizer was used for the left hand side of the window. Static boxesizers are the basic variations. GridSizers are important because they are contain graphs and charts. The bottom and top parts of the window have gridsizers. All of the rows and collumns must be the same size. FlexGridSizer sizers are grid sizers with the exception that the programmer can manipulate the dimensions. In this application a FlexGridSizer is used to display the information on the right hand side of the window. Another important aspect utilized was sashed windows. Sashed windows allow the window to be stretched making it larger or smaller. For example, if there are seperate pieces of information such as a graph and dataset, the sash enables the user to enlarge one of the the two to see more of the other. wxFormBuilder allows the programmer to create buttons and windows with ease. The program writes all of the 'cosmetic' code while the programmer creates the little working code.

The user interface was designed to be very intuitive. The process starts with importing data from a specified borehole. This data is placed in a chart at the top of the screen. It is placed with two columns on opposite sides of the chart. This data could be porosity, amount of radiation, or resistivity, depending on the data from the borehole. Once the data has been imported into the chart, the user chooses the method of interpolation. After the method is chosen, the semi-variogram is plotted on the PlotCtrlPane graph below the chart. There is a key on the right side of the graph. In the right vertical portion of the window are many options for formatting the graph. The user can select from three different mathematical models: gaussian, spherical, and exponential. Once the user chooses his model the program will generate the plausible points. The user can also choose the range and sill as well as the scale in the x and y direction. He can also choose the number of times the program will run. There is a tab for a label named "visual." In that window a graph created in OpenGL will appear. The graphic will correspond with different numbers. For example, red would represent a higher number while blue would represent a lower number. A small user guide has been

compiled to show this (Appendix C).

4.3 Computational

The overall structure of the code was dominated by the windowing, so the numerical code was worked in as methods. Rather than mixing it in with the GUI, all of these methods were collected into their own class. An instance of this data class is created upon the opening of a new project, opening file, or data import. Accessors are primarily used in setting the initial values from user input or file I/O, but are then mainly internal because they are only used in computations which are inside the data class. Computations, such as interpolation, are instigated by a call of the method in the frame class, and then the values can be accessed for display.

This approach of isolating the computational code is much cleaner and makes changes in the user interface easier since only method calls must be moved rather than blocks of code. Storing the data in a data class also allows it to be passed to other parts of the frame. The graphic pane needs the data to display, and C++ inclusion/dependency makes it difficult to get the values if they are an intergal part of the frame.

Details on the implementation on key methods in the computational section of the code is detailed in Section 3.

4.4 OpenGL Graphics

The data set has been represented visually using OpenGL, an interface to graphics hardware. OpenGL is complex and powerful, and is used for rendering interactive color images of three dimensional objects. There were two different types of views created: a height map and a three-dimensional terrain. Due to the inexperience of the team members with OpenGL, the programming guide and tutorials were heavily relied upon[4, 10]. OpenGL creates smooth, aesthetically pleasing images by automatically blending the programmed colors in the window. It also makes the sizing of the screen simpler. The distance between a coordinate point on the graph and the origin will stay in proportion during a change in size of the overall image. This allows the screen to be shrunken, grown, or put into full screen while keeping the picture the same, which is especially important in an application setting. There are many other options set in OpenGL: lighting, surface materials, fog, movement, etc. which are beyond what this project requires.

In order to access the data to be visualized, the pointer to the data class which stores that information is passed to the GLCanvasPane after the interpolation is completed. This prevents anything from being rendered until after the interpolation is finished, or if everything is filled, the “Go” button is pressed. The dimensions of the grid in number of cells and maximum and minimum values are also needed for rendering.

A terrain is a three dimensional surface with varying heights. It is created by having a two-dimensional mesh in the x-z plane and storing

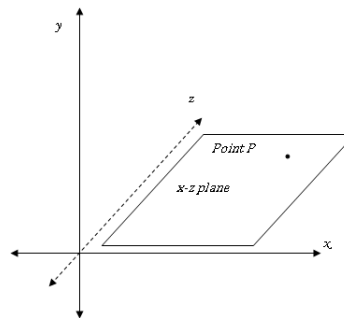


Figure 12: The x-z plane for the 2D grid. The y coordinates are set based upon the value at each point.

heights (y coordinates) for each point. Specific to this project, this means taking the two dimensional slice that is being modeled, and using the values in each cell to set the height. A method is defined to return a height for a given set of two-dimensional coordinates:

```
float GetHeight(int x, int z) { return ((data->GetValue(x,z)-min)/(max - min) - 0.5f); }
```

This takes the value relative to the minimum - which is the lowest point - and finds where it lies in the range. This returns a decimal, so subtracting 0.5 centers the object vertically. These heights are put out in the format of decimals so the programmer must convert them into values.

Besides setting the height, the color of the vertices are also set dependent on the value of each point. This was more complicated to do. First a color array is set up, ranging from blue to red. A red-green-blue (RGB) cube illustrates how the color changes from blue to green to red as different components are added and subtracted. The color array has structures for elements to store the red, green, and blue components of the colors. By setting the length of the color array proportional to the range of values, the index of a color can be calculated for a given value.

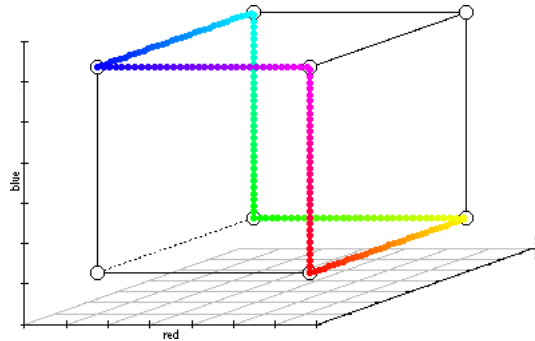


Figure 13: An RGB Cube - a graph of the colors with the red, green, and blue components on each axis.

$$\frac{index}{\#colors} = \frac{value}{max - min}$$

$$index = \frac{(value)(\#colors)}{max - min}$$

A method similar to the one for height is defined to return that index:

```
int GetColorIndex(int x, int z)
{ return ((data->GetValue(x,z)-min)*NCOLORS/(max-min)); }
```

The next step is to draw the actual surface. This is done with a triangle strip. A triangle strip takes given vertices and draws triangles for consecutive sets such as {1,2,3}, {2,3,4}, {3,4,5}. This is fast because there are fewer three-dimensional vertices that have to be sent to the graphics card. The method from the terrain tutorial was used for looping through the grid and setting the correct vertices. The routine used to render the terrain is as follows, with a sample result in Figure 14.

```
int z, x, index;
int sizing = 2.25;
float scale = 2.0f / MAX(width - 1, length - 1);
glScalef(scale, scale, scale);
glTranslatef(-(float)(width-1)/2, -0.5f, -(float)(length-1)/2);

for(z = 0; z < length - 1; z++) {
    glBegin(GL_TRIANGLE_STRIP);
```

```

for(x = 0; x < width; x++) {
    index = GetColorIndex(x, z);
    glColor3f(Rainbow[index].Red, Rainbow[index].Green, Rainbow[index].Blue);
    glVertex3f(x, sizing*GetHeight(x, z), z);
    index = GetColorIndex(x, z+1);
    glColor3f(Rainbow[index].Red, Rainbow[index].Green, Rainbow[index].Blue);
    glVertex3f(x, sizing*GetHeight(x, z+1), z+1);
}
glEnd();
}

```

The other type of image was taken from the idea of a heightmap. A heightmap is an image used to store three-dimensional data. It is essentially the two-dimensional plane with a range of grayscale to represent height. It is easier to see the range of data in this heightmap, so it has been included as a viewing option. The x-y plane is used in this case, and only the colors are set, not depth. This allowed the estimations of the unknown geological values to be shown in a window for easier intelligence. The values of the makeup of the land were given specific colors and then graphed in a window to create a two-dimensional model of the landscape.

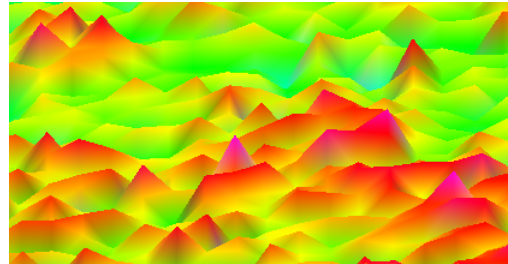


Figure 14: A cropped screen capture of a 3D terrain generated for sample data.

These graphics can be interpreted as red spots being the higher numbers with the other values colored accordingly. The preferable values and colors for finding an aquifer will differ between the graphs of different variables. In a graph of the porosity of the bore hole a higher number would be preferable to a lower because water will be retained in the rock more if the porosity is higher. However in a graph of gamma radiation a lower number would be better. This explains the lack of a key in the user interface window: making sense of the values is really up to the user.

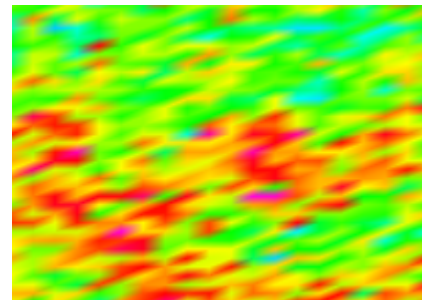


Figure 15: An example of the heightmap rendered for the same sample data as the terrain.

4.5 wxPlotCtrl Graphing

Even though there are several plotting packages available to use with wxWidgets, wxPlotCtrl is the one supported by wxFormBuilder. It is an interactive xy plot with options such as zooming, selection of points, and data processing. Many of these functions were hard to access and use owing to the lack of documentation. Despite this difficulty, an operational graph was created for the semi-variogram.

PlotCtrlPane is an object within the application frame. Once the semi-variogram is calculated,

the data is passed down to be plotted. The x and y directions are separate and distinguished by color in a key. The experimental semi-variogram is then interactively matched by the user. There are several options on the left sidebar. The data sets can be scaled to account for anisotropy by entering a percentage. This calls the function in the PlotCtrl which both performs the scale and saves the inverse. The next percentage that is entered will first use the inverse to revert back to the original size before applying the new scale. The sill/range of the semi-variogram is set by double clicking on the graph, which is interpreted by the library as coordinates. These are recorded in variables and used to calculate the mathematical models which are then added to the plot. These curves are created by setting the points every whole number, which can cause some misrepresentations for small horizontal sections. The sill and range can also be set originally from the sidebar. After they are set – either in the sidebar or with the mouse – the current values are displayed in the sidebar, updated in the idle loop, which prevents them from being set there again. Other methods of updating were attempted, but interclass communication and propagation of events has not been successful so far.

The wxPlotCtrl library also has some automatic functions that were useful. A click and drag of the mouse will zoom in on the selected section of the plot and can scroll along the axes. The title and labels can be edited, though they will always be reset.

5 Results

5.1 Case Study

This case study returns to the original application to aquifers: using data from boreholes to determine if water-bearing rock might be located in the ground between the holes. It is an ideal choice in some ways, since geostatistics was used in hydro-geology early on in its development. In most cases, it would be prudent to verify spatial dependence before interpolating, but for the limited scope of this project it has been assumed. This assumption is supported by the traditionalism of the field.

Regrettably, the two boreholes left to be used after one had to be dismissed were the farthest apart and proved too challenging for the present version of the program. In spite of these problems, progress has been made in completing the study and it has provided invaluable insights into what future work is required.

5.1.1 Data

Data from boreholes in the Los Alamos area were generously provided for use in this project (Section A.3). There are many different types of information collected from the boreholes; this project deals with depth, porosity, water flow, and radiation emission. While porosity or a different variable can provide valuable information about the locations of aquifers, it really is the *combination* of favorable qualities that will indicate a possible aquifer. This is because the perfect geological site for an aquifer is determined by many different variables acting together for the ideal surroundings. This

There was data for three boreholes in the Mortandad Canyon area (R-1, R-7, and R-33 located in Figure 17), but unfortunately, as it was prepared to be imported, it was realized that one did

not overlap with the other two depth wise and could not be used at this point. Borehole R-1 was eliminated and the other two were imported to test the program.

The same section of depth was observed by taking the elevations and depths for the boreholes. Even though the holes are not at the same elevation, it was possible to find an absolute measurement above sea level by decreasing the ground elevation by the depth of the hole at each point. This lined up sets of data, making it possible to take a consistent cross-section (Figure 16).

Five different sets of data were provided so that it would be relatively easy to write an import function to bring the sets into the program. The first set of measurements is standard Gamma Rays in API units. Gamma rays are used to find water and the different fluctuations in the rock patterns. The meters measure incoming gamma rays from radioactive elements in the surrounding rock. For example, if potassium or uranium are in a rock the water is most likely not there because they have a large nuclear signature. Since these elements are highly radioactive they will create quite a large signature and water will not be there because of lack of porosity. Even if there were porosity, it would be filled with the radioactive elements. The next type of data received was Deep-Reading Resistivity in ohm-meters. This basically reads the conductivity and resistivity of the rock at interval depths. The conductivity and resistivity of the different depths can be used to determine whether or not the rock holds any potential for water. The third set is a total porosity set which measures the total amount of pore space in the probed aquifer. This is a less effective way to find data because of its using all of the pore space no matter how small. More meaningful is the fourth data set, effective porosity. Effective porosity allows the probe to only find certain sized pores that allow effective flow of ground water measured by nuclear magnetic resonance (NMR). Last is the Logarithmic mean of T2. The Logarithmic mean is the average pore size of the probed area and can be used to tell how useful the aquifer is - the pores must be open enough to be interconnected.

An import method was then written to bring the data into the application and was then used as a sample set to test the program with real world data.

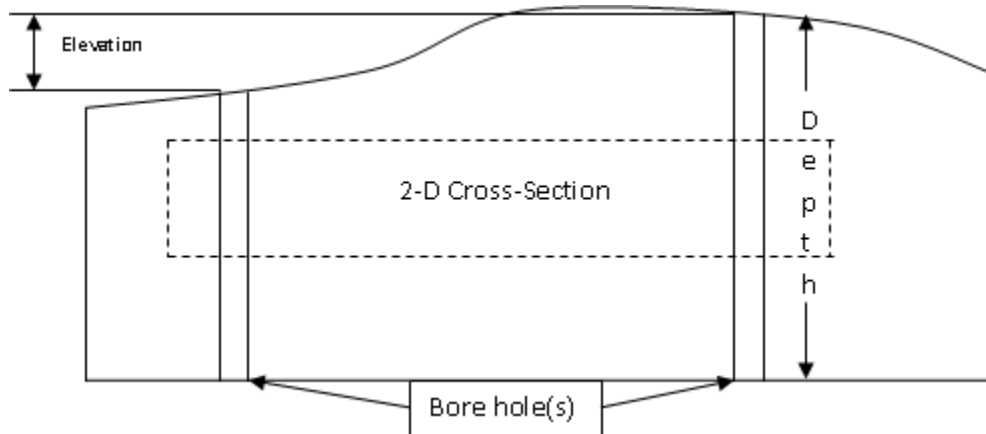


Figure 16: Two dimensional cross-section of boreholes with different elevations.

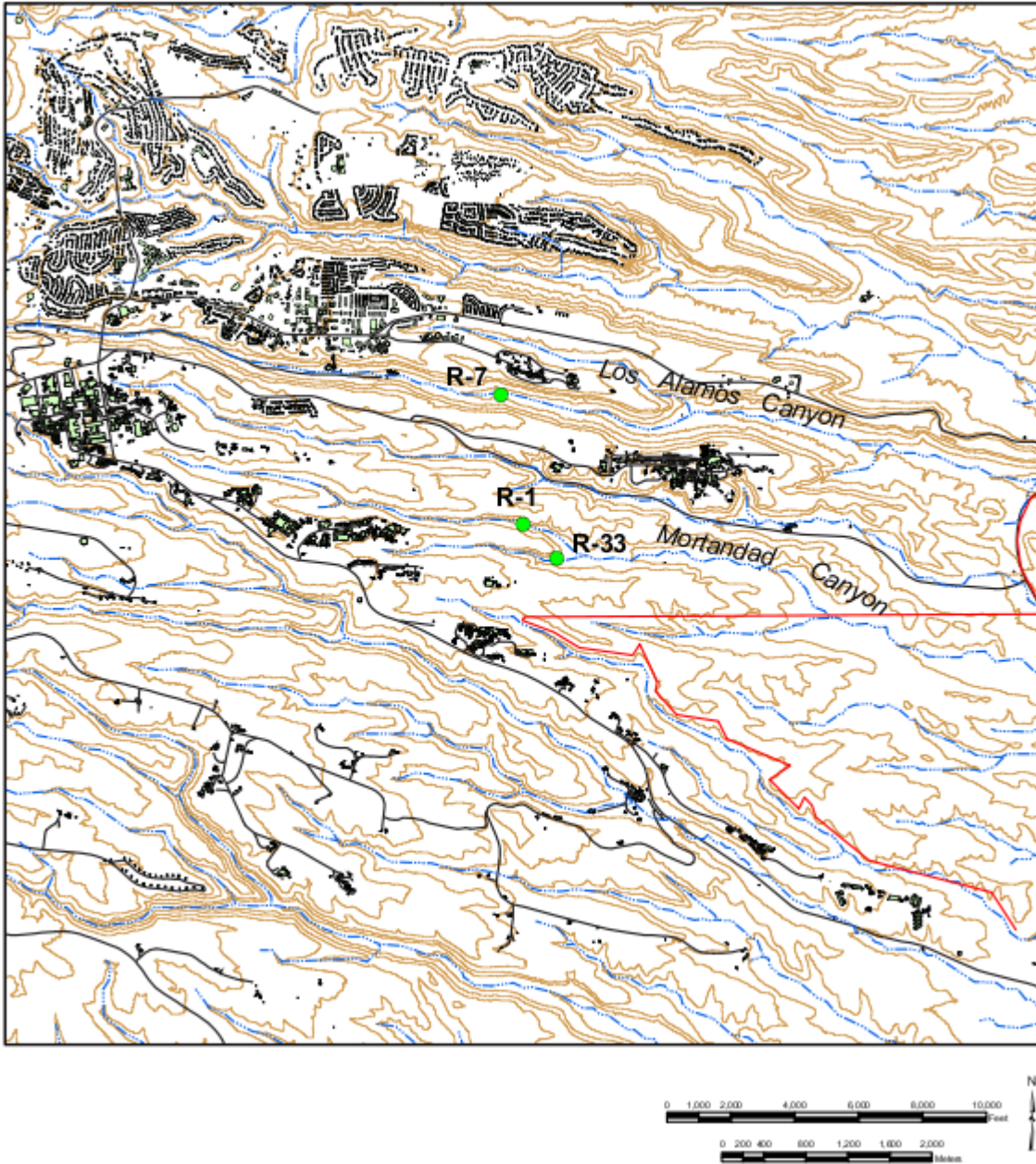


Figure 17: Location of the boreholes from the case study.

5.1.2 Semi-variogram

The semi-variogram for the various types of data was successfully plotted. The x direction has only one point because there is only the single distance from one borehole to the other. The y direction produced interesting plots which are included here in Figure 18. Note the large values of variance that probably arise when the data covers multiple rock structures. The large differences between them would increase the average variance.

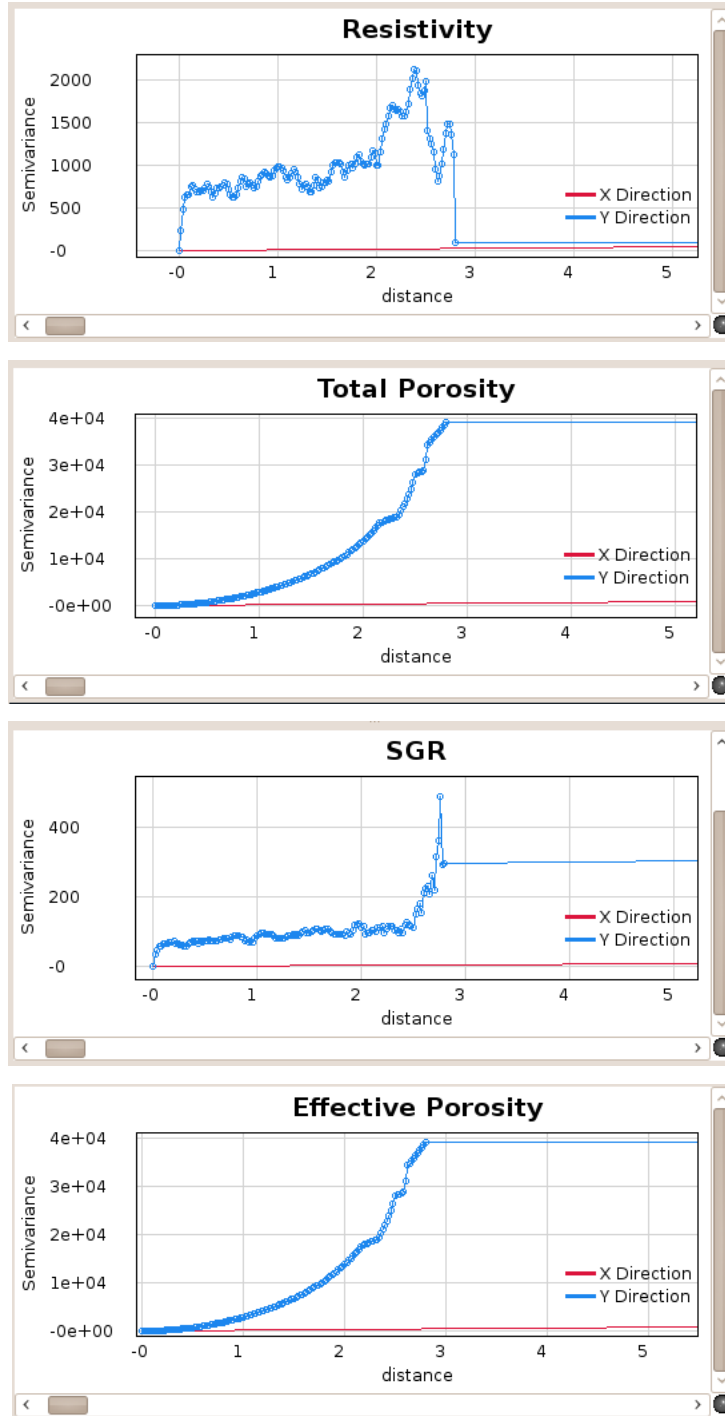


Figure 18: Semi-variogram of each of the data sets for the two boreholes. These are focused on the y direction as there is only one known point in the x direction.

5.1.3 Mixed Success of Interpolated Fields

There were mixed results between inverse distance weighting and kriging. The IDW was able to complete the interpolation, while kriging returned completely unreasonable numbers. These runs used the anisotropy and sparse data options and only run for each since the problem is so large.

The height maps produced for each data set using IDW are shown in Figure 19.

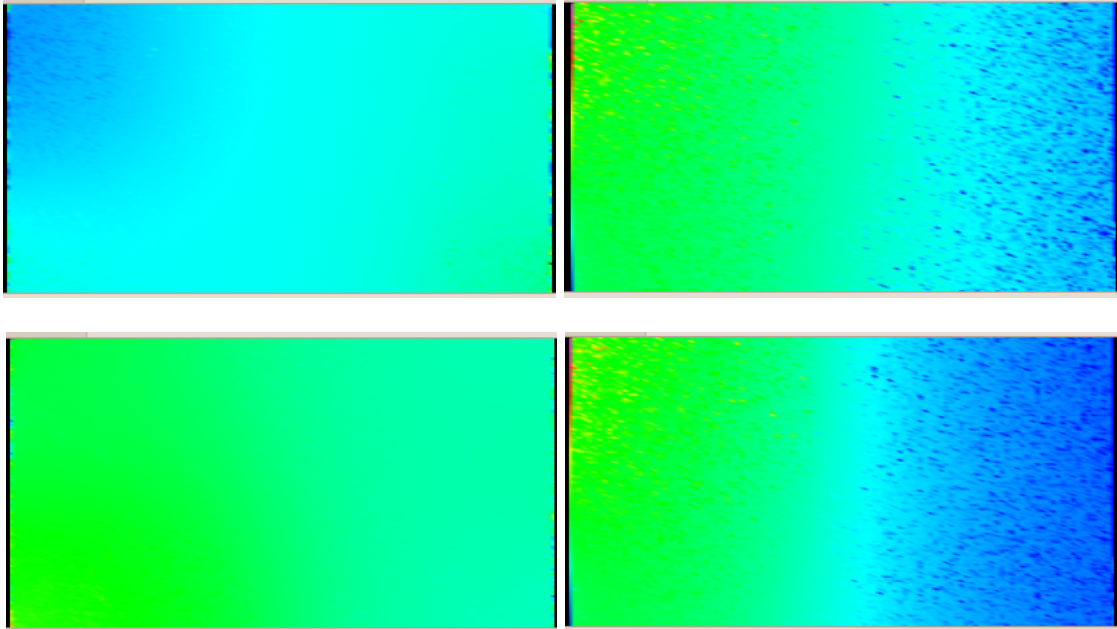


Figure 19: Height maps produced for inverse distance weighting interpolation between boreholes. From top to bottom they are: resistivity, total porosity, SGR, and effective porosity.

Since these results were only obtained at the end of the project, no attempt at numerical analysis has been made.

Attempts to run the borehole problems using kriging continued to be fruitless. Even though it was finally capable of finishing, the numbers were ridiculously large. In interpolation, results should fall within the range of the original data, further invalidating those values. It was theorized that the large variances used in matching the semi-variogram, coupled with the scale of the problem, simply overloaded the methods and caused it to return nonsense. Ultimately, a small section of the boreholes was used instead, and it was successfully completed when an much smaller variance was used (Figure 20).

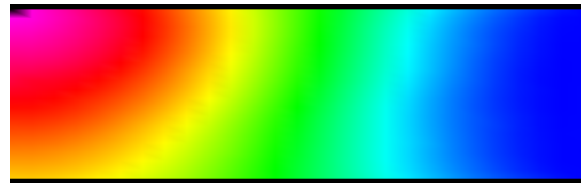


Figure 20: Visual results of small section of borehole interpolated with kriging.

6 Conclusions

There was a lot accomplished in the course of this project, but geostatistics is a complex field, so only the fundamental levels were covered in the available time. The science that was managed to be incorporated into the project is potentially useful in the a search for aquifers, as shown by the preliminary results, and was a significant learning experience for all involved.

The results of the case study demonstrated that there are still several adjustments to be made

for the program to be a viable tool in large scale problems. For smaller problems, it works well. This includes several options set by the user for desired effects. The user interface still has some pitfalls if it is not used the way it was designed, but turned out impressively.

Interesting characteristics of the two interpolation methods became apparent through the test runs. Kriging provides a smoother interpolation in contrast to the 'bull's eye' effect of inverse distance weighting. There are advantages of IDW which were demonstrated in the case study. Kriging is much more sensitive to the high variances and anisotropy, which likely caused the faulty results when it was used for the borehole data. IDW can complete this problem, returning results for similar situations that are not possible with kriging.

More than the cursory attention given to the results from the case study would be required to determine exactly how accurately the program predicted possible aquifers. Unfortunately, these were not completed until the very end of the project, so this analysis was not possible. In and of itself, simply getting plausible results out of the program is an achievement.

6.1 Current Status

Despite hard work and good ideas there were many problems in the making of a working program. There was a memory leak that made the system crash if overly extensive data sets were attempted to be run. The memory leak was fixed, but large data sets still do not work, returning very large, faulty answers. Smaller data sets do run through the program and yield successful results. The case study involving the borehole data was imported into the program and helped to determine whether the process worked. To hasten the slow computation of the computer program an optimization of tasks was attempted. It was partially successful and made the program somewhat more efficient.

The graphics presented many difficulties and much time was spent in fixing them. Getting them to interface with the C++ was problematic and did not work for a long time. Making the graphic map three dimensional was a trial. Initially the graphics were being written in OpenGL as rows of colored boxes that would be colored by their specified values, however that could not be contrived to work correctly and was discarded in favor of triangle stripping. This worked much better and the graphics finally interfaced satisfyingly with the other code.

Fortunately, almost all of the problems encountered in the progress of "Adam's Ale" were dealt with and eliminated to create a program that does what it was designed for. Hopefully, it will be able to solve real world problems.

7 Teamwork

When a team has a small number of members, the confines of the duties are less well defined, as more work has to be done by fewer people. Team 65 had four members of whom each had a very specifically designated task. This system was designed for efficiency and effectiveness and was implemented with success. As well as giving each job to the most suitable team member, Team 65 attempted to involve each of their colleagues in their own allocated areas so as to provide instruction for everyone. The team members not already versed in programming learned the basics of OpenGL, wxFormbuilder, C and C++. Because of the program's application in geostatistics pertaining to aquifers, everyone mastered the information on aquifers needed to make this project a success.

One of the many ways Team 65 improved on communication was through the utilization of Google Code, Google Documents, and Google Calendar. Google Code granted the members of the team working on programming a repository for their specialized parts of the whole. The repository made the merging of the parts a quick and easy process. Google Documents was a repository for the members of the team working on the various reports so that the many different parts of the writing could be done more easily and the other teammates could immediately have the most current versions of the reports. The individuals of Team 65 are not especially noted for their organizational skills, however Google Calendar rectifies this matter. It could be said that having their schedule of events available for regular perusal impressed upon them the responsibilities of remembering a meeting. Not only would the other teammates be disappointed in the individual if he missed but would exclude him from the amusing antics Team 65 would routinely engage in while working diligently.

Overall, Team 65 functioned under very superior working conditions for the entirety of their project and were extremely pleased in the end result of all their hard work.

8 Recommendations

This project created a solid basis for future studies in the extensive and growing field of geostatistics. The discontinuity of success from small to large problems should be addressed, perhaps in the form of dividing the problem up into the separate structures. Additions of safe-guards to catch errors that may occur for incorrect user operations would prevent unexplained crashes. The user should be supplied with helpful messages to guide them to fix the problem in the input.

Efficiency in a computer program is always to be desired and optimization is the key to a faster running program and should definitely be undertaken to better the program. The operation on the GPU is still visibly slower than the CPU and requires more work to get any speed-up. This should be done by parallelizing more sections of the method and balancing the gain of the parallel processing with the cost of loading the data.

The case study involved in the program introduced the issue of creating an accurate semi-variogram as only one value could be calculated in the x direction - at the distance between the boreholes. It seems that this is still a dilemma of the field and that the graphing of semi-variograms from sparse data has yet to be performed in the field of geostatistics. If this project managed to achieve this milestone then many more could follow.

These improvements could take this project to a more complex and hopefully more useful level of practice.

A References

A.1 Bibliography

References

- [1] Gaussian function. *Wikipedia*, 2009.
- [2] Isobel Clark. *Practical Geostatistics*. Elsevier Applied Science, 1979.

- [3] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, Maryland, 3rd edition, 1996.
- [4] Bill Jacobs. Opengl video tutorial - terrain. 2008.
- [5] Shari Kelley and Peggy Johnson. Frequently asked questions about water. *The New Mexico Bureau of Geology and Mineral Resources*, 2009.
- [6] G. Matheron. The theory of regionalised variables and its applications. Technical report, 1971.
- [7] NVIDIA. *OpenCL Programming Guide for the CUDA Architecture*, 2.3 edition, Mar. 2009.
- [8] Howard Perlman. Water science for schools: Aquifers. *United States Geological Survey*, 2009.
- [9] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, New York, 1988.
- [10] Dave Shreiner. *OpenGL Programming Guide*. Addison-Wesley Professional, Upper Saddle River, New Jersey, 2009.
- [11] Julian Smart, Kevin Hock, and Stefan Csomor. *Cross-Platform GUI Programming with wxWidgets (Bruce Perens Open Source)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.
- [12] Kimberly J. Swanson. Aquifer characteristics. *Water Encyclopedia: Science and Issues*, 2009.

A.2 Software/Tools

Several programming tools and other applications were used in the development of this project:

- wxWidgets/wxFormBuilder
- Eclipse
- Totalview
- Doxygen
- Google Code
- Microsoft Office and OpenOffice
- LyX

A.3 Acknowledgments

We would like to extend our deepest thanks to the people who volunteered so much of their time to help us with this project:

- Robert Robey, for his general help with everything;
- Thomas Robey, for his mentorship in the geostatistics and mathematics;
- David Broxton, Danny Katzman, and Ned Clayton of Schlumberger, Inc for providing us with data, definitions, and meeting with us to work through it;
- Mary Green, for her advice in geology and hydrology;
- Larry Cox and Jorge Crichigno and for their advice and positive comments at the interim presentation.

B Glossary

Anisotropy property of directional dependence; in geostatistics this means that the data has different spatial correlation in the x and y directions. For example, wood will be more related along the grain than against it.

Geostatistics a branch of applied statistics that uses the interdependence of spatially correlated data to interpolate unknown values.

Isotropy property of consistency for all directions, in geostatistics the spatial correlation is independent of direction.

Kriging a common method of interpolation in geostatistics that uses a mathematical model of the semi-variogram.

Nugget (of semi-variogram) magnitude of discontinuity at the origin, usually a result of measuring/sampling errors.

Range (of semi-variogram) distance at which the semi-variogram plateaus and range at which points are correlated to some degree.

Semi-Variogram the formula and graph of the variation of data over distance, quantifying spatial correlation.

Sill (of semi-variogram) variance value for distances beyond the range or the value of the plateau

Spatial Correlation the idea of data being related as a function of its location

C User Guide

This program was created to be very user friendly. The window is a very simple design in which there are three large sections. The first section is a tabbed window in which the user can switch between the visual and the raw data. Below that is the semi-variogram. That is the area in which the data is graphed. On the right side is a vertical window in which the user can change the appearance of the semi-variogram. For example, the user can choose the range, sill, a model, the method of interpolation and the size at which the X and Y values are scaled.

-To start the process, either go to File->Open... or select File->New. Then select the set of data that is to be implanted or created.

-It will open a window that has the uploaded data; select the set of data.

	A	B	C	D	E	F
1	1.250200	0.068800	0.335200	0.897900	-0.966700	-0.0572
2	-0.855300	0.997300	0.628000	-0.884000	-1.270800	-0.5100
3	0.493500	0.583500	-0.535700	-1.923500	-0.652300	-0.9452
4	0.533300	0.092100	1.846300	0.372100	0.764600	0.61400
5	0.171100	0.712200	0.194000	0.532800	-0.353300	-0.9345
6	1.016900	0.469400	0.588400	1.874600	0.267800	-0.8214
7	1.379000	1.501300	0.227500	-0.193500	0.545800	0.130700

Figure 21: Data inserted

- Once that data set has been opened the chart at top of the window that reads “Raw Data” will fill with the selected set of data.
- Plot->Semivariogram, the graph will appear in the bottom section.
- Double click on the knee of the graph. Three more lines will appear, they are the three different mathematical models.

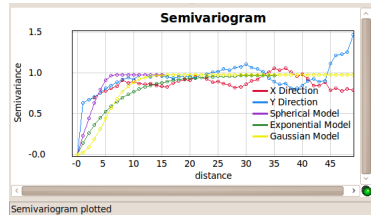


Figure 22: Semivariogram plotted, mathematical models chosen

- Select one of the models at the right in the “Model Type” scroll box. The model which has been selected will now be the only one on the graph.
- Choose your method of interpolation, the scale for each X and Y direction, and the number of runs.

Figure 23: Model type, range and sill, scales, interpolation method, number of runs selected

- Hit “Go!”.
- There are many paths the user can take from this point.
- It is possible to view the data in which the semivariogram has “come up with”.

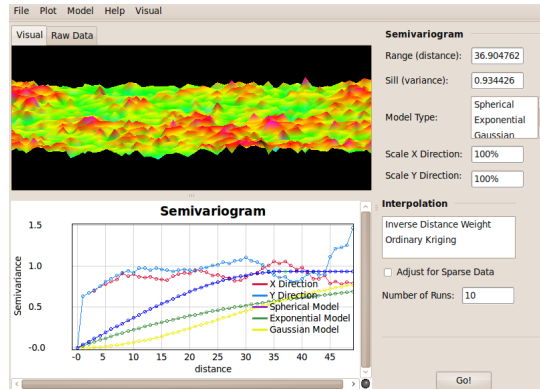


Figure 24: Final screen with a terrain map

-Also, the user can view two different versions of the visual with the “Visual” drop down menu. Choose between “Height Map” and “Terrain”.

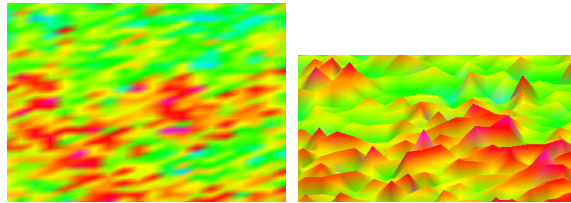


Figure 25: Left->Height Map, Right->Terrain

To Kill a Flocking Bird

New Mexico
Supercomputing Challenge
Final Report
April 6, 2010

Team 70
Los Alamos High School

Team Members

Peter Ahrens
Stephanie Djidjev
Vicky Wang
Mei Liu

Teacher

Lee Goodwin

Project Mentors

Christine Ahrens
James Ahrens

Table of Contents

TO KILL A FLOCKING BIRD	1
<hr/>	
TABLE OF CONTENTS	2
1.0 EXECUTIVE SUMMARY	3
2.0 STATEMENT OF THE PROBLEM	4
3.0 DESCRIPTION OF THE METHOD USED TO SOLVE THE PROBLEM	5
3.1 THE NETLOGO FLOCKING MODEL	5
3.2 EVALUATION AND GOODNESS FUNCTIONS	6
3.4 BRUTE FORCE PARAMETER STUDY	10
3.5 OTHER SEARCH METHOD IMPLEMENTATIONS	10
4.0 RESULTS	12
5.0 CONCLUSIONS	15
6.0 SIGNIFICANT ORIGINAL ACHIEVEMENT	16
7.0 WORK PRODUCTS	16
7.1 FLOCKING WITH GOODNESS FUNCTIONS	16
7.2 BRACKETING	17
7.3 STEEPEST DESCENT	17
7.4 GENETIC	18
8.0 BIBLIOGRAPHY	18
9.0 ACKNOWLEDGEMENTS	19

1.0 Executive Summary

This project explores which search techniques work best to optimize the parameters of a flocking model. Flocking is a natural phenomenon of many independent agents (birds) making decisions that lead to the group acting as a whole. The parameters used to control flocking are the angle at which a bird turns to get closer to his neighbors, the angle at which a bird turns to align itself with the rest of the flock and the angle at which a bird turns to get away from his neighbors if he is too close. NetLogo was used to develop an algorithm to judge qualities of a flock, implement the search techniques, run the search techniques and gather the data for comparison. The search techniques used were brute force (a test of all the possible combinations of parameters), genetic algorithms (a random search variant modeling natural selection), bracketing (dividing the search space iteratively), and steepest descent (searching locally and proceeding in the most promising direction to the solution from a random starting point in the search space). To evaluate a flock, a goodness function was created from the following functions: average distance to center, average difference in birds' distance to center, the average difference in the spacing of each bird to its nearest neighbor, and the average difference the birds' headings. A visual analysis of the brute force parameter study showed a diagonal gradient through the search space. The other search methods were tested, and compared based on the quality of the flocks produced, the reliability of the search, and the time efficiency. The results showed that the steepest descent technique had good performance and produced the best result.

2.0 Statement of the problem

Flocking is a natural phenomenon of many independent agents making decisions that lead to the group acting as a whole. Some examples of flocking behavior happen frequently in nature and they serve different purposes. Fish may exhibit flocking behavior to make themselves look bigger and ward off predators. Birds exhibit flocking behavior when they migrate. They rotate out of the front position in the flock and thus conserve energy breaking the wind. Elephants also exhibit flocking behavior for a different purpose. The larger elephants form a ring around the smaller, weaker elephants in an attempt to keep them safe from predators. Their flocks do not move much.

Flocking occurs in the manmade world as well. Flocking can be seen in strategic military formations and it can also be seen in traffic patterns (people tend to follow one another on large freeways). The principles of flocking can be applied to collision detection in robotic domains. Robots are usually programmed to avoid other robots or obstacles (unless they are battle bots). Flocking principles can also be applied to military applications with computer driven vehicles.

An interesting thing about flocking is that a computer can model it. Each decision-making entity, an “**agent**”, begins in a random position, then using the location of its neighbors, makes decisions as to where to move itself. These decisions are usually called **cohesion**, **alignment** and **separation**. To cohere, an agent will move itself closer towards its neighbors. To align, an agent will align its heading with that of its neighbors. To separate, an agent will move itself away from its neighbors if it is too close. If these decisions are carefully balanced, the agents in the model will form a flock after a number of time steps. Balancing these decisions is a challenge and the flock quality is directly dependent upon the balance. Measuring the quality of the flock is a subjective process. Not everyone will agree that the quality of a flock is the same.

Usually the flocking decisions cohere, align, and separate are the parameters of the flocking model. How would you find the best parameters? The only certain way to do this would be to

test them all in all their combinations (thousands), and evaluate the resulting flocks using a function that tells you the quality of the flock or if it is a flock. This is a **brute-force** method of optimizing the parameters. This is computationally intensive and inefficient in its use of time, but it is the most accurate way of finding the best parameters.

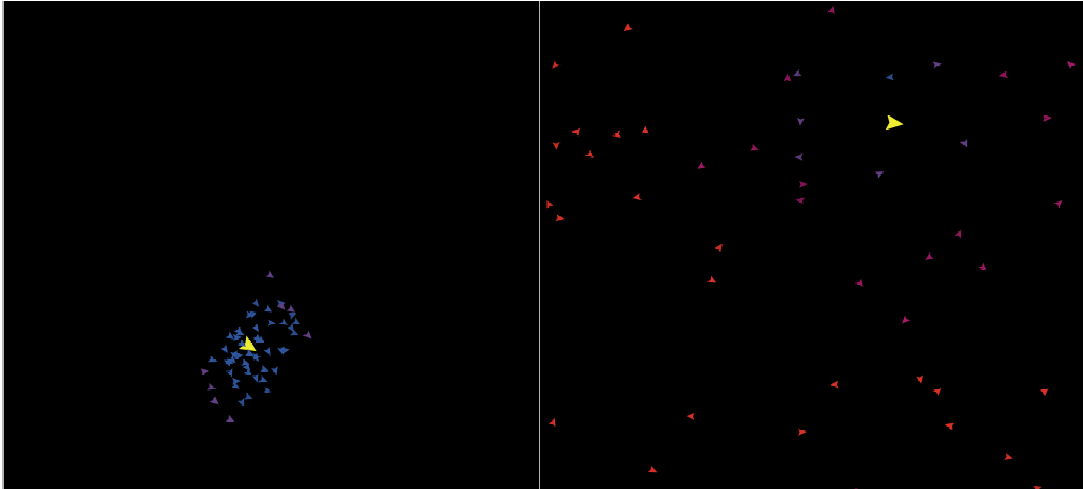
Many common search techniques used widely in computer science can be used for finding parameters, however not all of them are best suited towards flocking. Flocking can be unpredictable and time-intensive to search through and thus not all search techniques will perform the same. This project aims to find out which common search techniques will perform the best, be the most accurate and be most reliable in finding the optimal parameters to a flocking model. Others who are building or using flocking models can use this research.

3.0 Description of the Method Used to Solve the Problem

The method used to solve the problem of finding the best parameter search method started with a simple flocking model. This model was modified the model to evaluate the “goodness” of the flock by using **goodness functions** created by the team. After testing the goodness function visually, a brute-force method was used to understand the search space. Three search methods were then developed and tested, to find which produced the best parameter combination. The brute-force and other search methods were compared using statistics.

3.1 The NetLogo Flocking Model

A NetLogo model of flocking was found in the sample models included with NetLogo. NetLogo® was chosen because it is perfectly suited to flocking models. NetLogo® is an agent-based program and it is iterative. There is a graphical user interface and a display that shows the agents flocking and the time steps so far. The original code did not include any plots and all the parameters were controlled by sliders (the user). There were no flock quality evaluation functions included. In this model, the agents look like birds and flocked in a direction (they are moving as they flock), which is similar to the way that birds flock. The agents will be referred to as birds hereafter.



Picture 1: On the left, a flocking behavior is seen, on the right, there is no flocking.

The parameters used to control the decisions the birds made in this model **are max-cohere-turn, max-align-turn, max-separate-turn and minimum-separation**. Max-cohere-turn designates the maximum angle a bird can turn to cohere. It is the same with max-align-turn and max-separate-turn; they are also the maximum angle a bird can turn to make a flocking decision (align or separate). Minimum-separation is the distance between birds that signals that they need to turn away from each other. Originally, minimum separation was thought a parameter to be optimized, but was then understood to be a preference, as it designates how big a flock will be, which does not affect flock quality.

3.2 Evaluation and Goodness Functions

The first step was to develop a way to evaluate the flock. The goal was to have one goodness function that would tell how good the flock was. The goodness function used turned out to be an average of several evaluation functions. The evaluation functions developed were **mean distance to center, mean deviation of agents' distance to center, the mean deviation in the spacing of each agent to its nearest neighbor, and the mean deviation of the agents' headings**.

In order to evaluate the goodness of the flock, it was necessary to calculate the center of the flock. Although it was initially difficult to calculate the center of the birds in a boundless domain, a center of mass algorithm [7] was used. Using this algorithm, a **center bird** was created. It is enlarged for better visibility. Since it is the biggest bird of them all, it is colored yellow in honor of Big Bird. The center bird represents the center of the flock and its heading is the average heading of the flock.

The first evaluation function we developed was quite simple. It was the mean distance to center, which is exactly what the name implies. It is the average of all the distances from each bird to the center bird.

$$\text{mean distance to center} = \frac{\sum_{n=0}^{pf} \left(\sqrt{(x_n - x_c)^2 + (y_n - y_c)^2} \right)}{pf}$$

Equation 1: Where pf is the population of the flock, x_n is the x-coordinate of the n^{th} bird, x_c is the x-coordinate of the center, y_n is the y-coordinate of the n^{th} bird, and y_c is the y-coordinate of the center.

It indicates how clustered the birds are. A low value means the birds are clustered close to the center. A high value means they are spread out over the whole domain. A zero value indicates the birds are all in the same position (in a dot). This function was not used in the final code, because if the model is optimized completely to this function, the birds will be centered in a tiny dot.

The second evaluation function developed was the mean deviation of the birds' distance to center. This evaluation function measures the deviations in the distances between each bird and the center bird. This measurement will show how spread out and randomly spaced (within the flock) the birds are in relation to the center and each other.

$$\text{mean deviation of distance to center} = \frac{\sum_{m=0}^{pf} \left(\text{abs}(\sqrt{(x_m - x_c)^2 + (y_m - y_c)^2} - \text{mean distance to center}) \right)}{pf}$$

Equation 2: Where pf is the population of the flock, x_m is the x-coordinate of the m^{th} bird, x_c is the x-coordinate of the center, y_m is the y-coordinate of the m^{th} bird, and y_c is the y-coordinate of the center.

A low value indicates less deviation, which means the birds are more ordered. A high value indicates clumping (multiple clusters of birds) and a non-optimal flock. A zero value (theoretical max optimization) means the birds are either in a circle centered around the center bird or all at the center point, but since this is usually never achieved, it make this an overall useful evaluation function.

The third evaluation function developed was the mean difference in heading. This is what the name implies. It measures the average deviations in each birds' heading compared to the average flock heading.

$$\text{mean heading deviation} = \frac{\sum_{n=0}^{pf} \left(\arctan \left(\frac{\sum_{i=0}^{pf} \sin(h_i)}{\sum_{j=0}^{pf} \cos(h_j)} \right) - h_n \right)}{pf}$$

Equation 3: Where pf is the population of the flock, h_n is the heading of the n^{th} bird, h_i is the heading of the i^{th} bird, and h_j is the heading of the j^{th} bird.

While this function does not exactly measure how good a flock is spatially, it does show that the flock is not bumping into each other or going different directions. A low value indicates the birds are all heading in the same direction. A high value indicates the birds are running into each other in a central flock, not flocking, or going different directions. A zero value (theoretical max

optimization) means they are all going the same direction exactly, which does not necessarily indicate a good flock, but it does mean the birds are not running into each other.

The fourth evaluation function developed is the mean difference in spacing between each bird and its nearest neighbor.

$$\text{mean spacing distance} = \frac{\sum_{l=0}^{pf} \left(\sqrt{(x_l - x_{nn})^2 + (y_l - y_{nn})^2} \right)}{pf}$$

Equation 4: Where pf is the population of the flock, x_l is the x-coordinate of the l^{th} bird, x_{nn} is the x-coordinate of this bird's nearest neighbor, y_l is the y-coordinate of the l^{th} bird, and y_{nn} is the y-coordinate of the birds nearest neighbor.

$$\text{mean spacing deviation} = \frac{\sum_{q=0}^{pf} \left(\text{abs} \left(\sqrt{(x_q - x_{nn})^2 + (y_q - y_{nn})^2} - \text{mean spacing distance} \right) \right)}{pf}$$

Equation 5: Where pf is the population of the flock, x_q is the x-coordinate of the q^{th} bird, x_{nn} is the x-coordinate of this bird's nearest neighbor, y_q is the y-coordinate of the q^{th} bird, and y_{nn} is the y-coordinate of the birds nearest neighbor.

It measures the mean difference in the distances between these birds and the birds closest to it. This function is very effective at measuring even spacing. A low value indicates the birds have evenly spaced themselves in relation to each other. A high value indicates the birds are either clumping or not flocking. A zero value indicates the birds are in an isometric dot pattern or a dot in the center of the screen.

All of these evaluation functions have shortcomings if used exclusively, but if averaged, they produce an accurate measurement of the quality of a flock. The average of these evaluation functions is our goodness function. After time was spent studying the effectiveness of the functions visually, the functions were weighted at the values in Table 1.

Evaluation Functions	Weight
Mean distance to center	0.7
Mean deviation in the distances to center	0.75
Mean deviation in spacing between birds	1.0
Mean deviation of the agents' headings	1.0

Table 1. Weighting of evaluation functions within goodness function.

3.4 Brute Force Parameter Study

Before any search methods were run, a brute force search was done to understand the search space better. To do this, a NetLogo tool called “Behavior Space” was used. It is a tool designed to run parameter searches and similar tasks. It ran the flocking code for 200 iterations for each combination of the input parameters on a thirty-bird flock. For minimum-separation, the value was a constant 0.75. Note that minimum separation is just a preference for how big the end flock should be. For a 30-bird flock, .75 is a sufficient minimum separation, accounting for a 72-square-unit domain. For max- {cohere, align and separate}-turn they were increments of 1 between 0 and 10. Those were the original ranges offered by the interface to the original flocking code and anything outside that range produces a bad flock. Even though the parameters for the angles could be from 0 to 180, the goal is to optimize between 0 and 10. Each parameter combination was run once. The Behavior Space tool output a comma-separated file containing which combination of parameters was used and what the goodness function value was for that combination of parameters. This file was read into Microsoft Excel, edited to remove irrelevant data, saved as comma-separated file again, renamed to a “.particle” (ParaView compatible) file and read into ParaView [11] for analysis by visualization.

3.5 Other Search Method Implementations

Three parameter search methods were to be used: **bracketing**, **steepest descent** and **genetic algorithms**. These search methods were chosen because they are some of the most widely used and applicable to flocking.

A framework was used in all three optimizations to make the search methods comparable and for code reuse. The framework consisted of the general parameter search steps: **generate**, **evaluate** and **select**. The generate step takes in a tuple of parameter combination and parameter bounds (call it a “**state**”) and generates multiple states to be tested. These states are tested in the next step, evaluation. To evaluate, the flocking model is run for a set number of iterations and the result of the goodness function is coupled to each state. Finally, in the select step, the state with the best goodness function is selected and if the simulation is allowed to continue, it is fed back into the generate step, otherwise, this last state is the output. Each search method has its own stopping criteria.

The first parameter search technique implemented was bracketing [9]. This technique divides the search space in half and finds the best half, then uses the best half as the starting point for the next iteration, until the remaining half is small enough. In the generate step of bracketing, for each parameter, the min and the max were averaged to produce Point B. The average of the min and Point B, Point A, was calculated and the average of Point B and the max, Point C was calculated. A list of all the possible combinations of Point A and Point C for all four parameters is generated. To evaluate, the simulation is run for a specified number of iterations and the goodness function result is coupled with the parameter combination used. To select, the parameter combination with the lowest goodness function value is selected. For each parameter, if Point A was better, then the max is set as Point B. If Point C was better, the min is set at Point B. These min and max values are used for the next iteration. The stopping condition developed for this search technique was that after four iterations of gen-eval-select, it would stop. After four iterations, the values were precise enough for the parameter range used (0 to 10 for each parameter).

The second parameter search technique implemented was steepest descent [8, 9]. This technique starts at a random point in the search space and evaluates the local surrounding search space, then proceeds one step towards the most promising direction. To generate in the steepest descent parameter search method, all the possible combinations of each parameter value being incremented one step up or one step down are generated. Step size was set at .05 times the max value for that parameter. Each of these combinations is evaluated like in bracketing and each

combination is coupled with its goodness function value. To select, the best combination is sent back as input to the generate step. It stops when it generates a flock with a goodness function value under 0.1. This is the value deemed “good-enough.”

The third search technique implemented is a genetic algorithm [1, 8]. It works by considering parameters to be genes that can be mutated, starting with organisms with randomly generated genes, evaluating them and choosing the organism with the best value to live, and others to die and be replaced by a mutation of the genes of the best organism. The generate step takes a pre-selected list of the worst in the flock, changes their parameters to be that of mean of the best birds and applies a mutation to that. The evaluate step runs like in the previous search techniques. To select, the best in the flock are selected and sent on to the generate stage. It stops when it generates a flock with a goodness function value under 0.1. This is the value deemed “good-enough.”

4.0 Results

The purpose of the experiment was to find the optimal parameter search technique for flocking. Therefore, a brute force study was run to understand the search space. Subsequently, the various parameter search techniques were run, evaluated and compared.

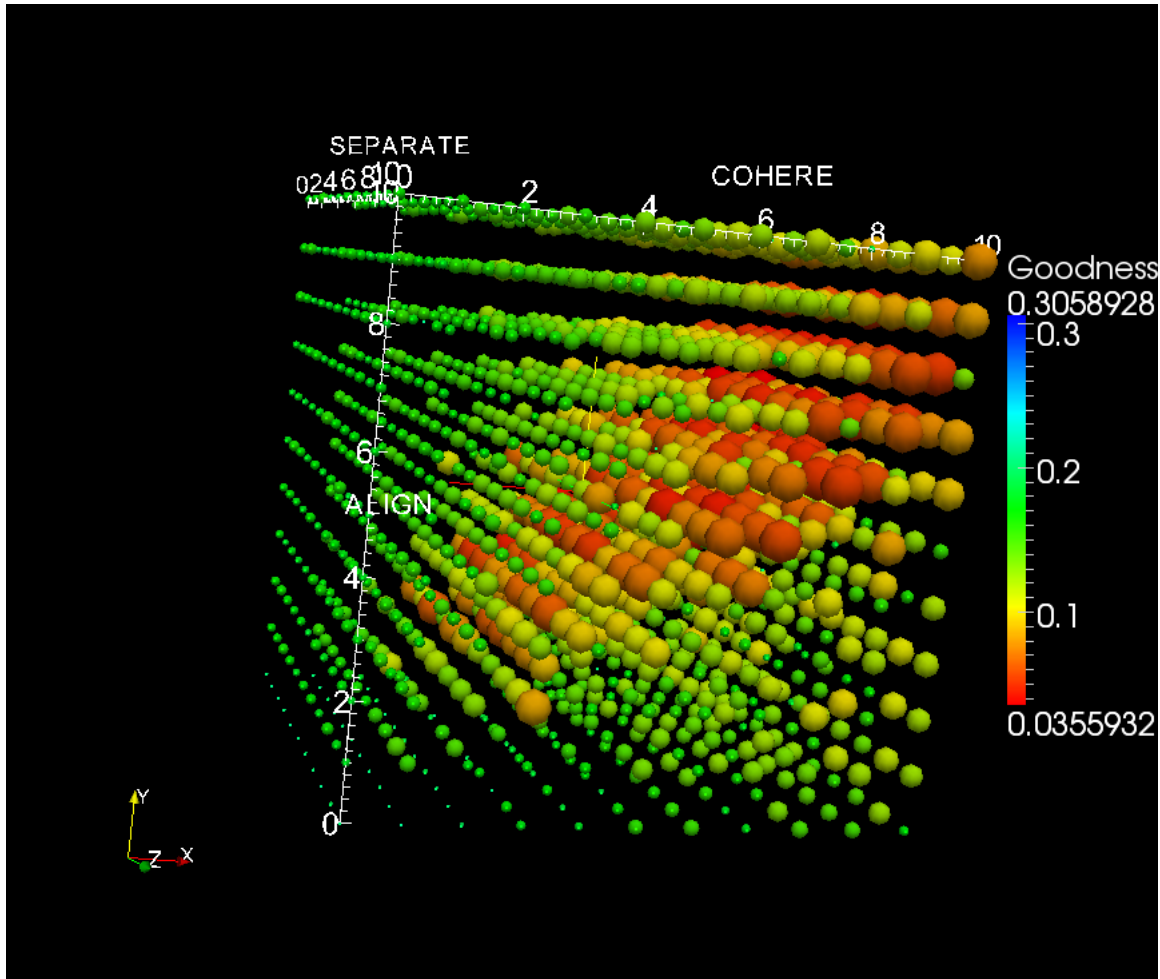


Figure 1. The parameter search space discovered via brute-force. Sphere radii and color represent goodness values.

In Figure 1, the results of the brute-force parameter study are shown. The radii of the spheres are inversely proportionate to the goodness value (the lower the goodness value, the better the flock, the bigger the spheres the better the flock). Different parameters are shown on each axis (cohere, align and separate). The colors also signify goodness, with red being the best flock, blue being the worst. The figure shows that when cohere and align are approximately equal in value, a better goodness value is found. From this figure, a parameter search space with a diagonal gradient can be seen.

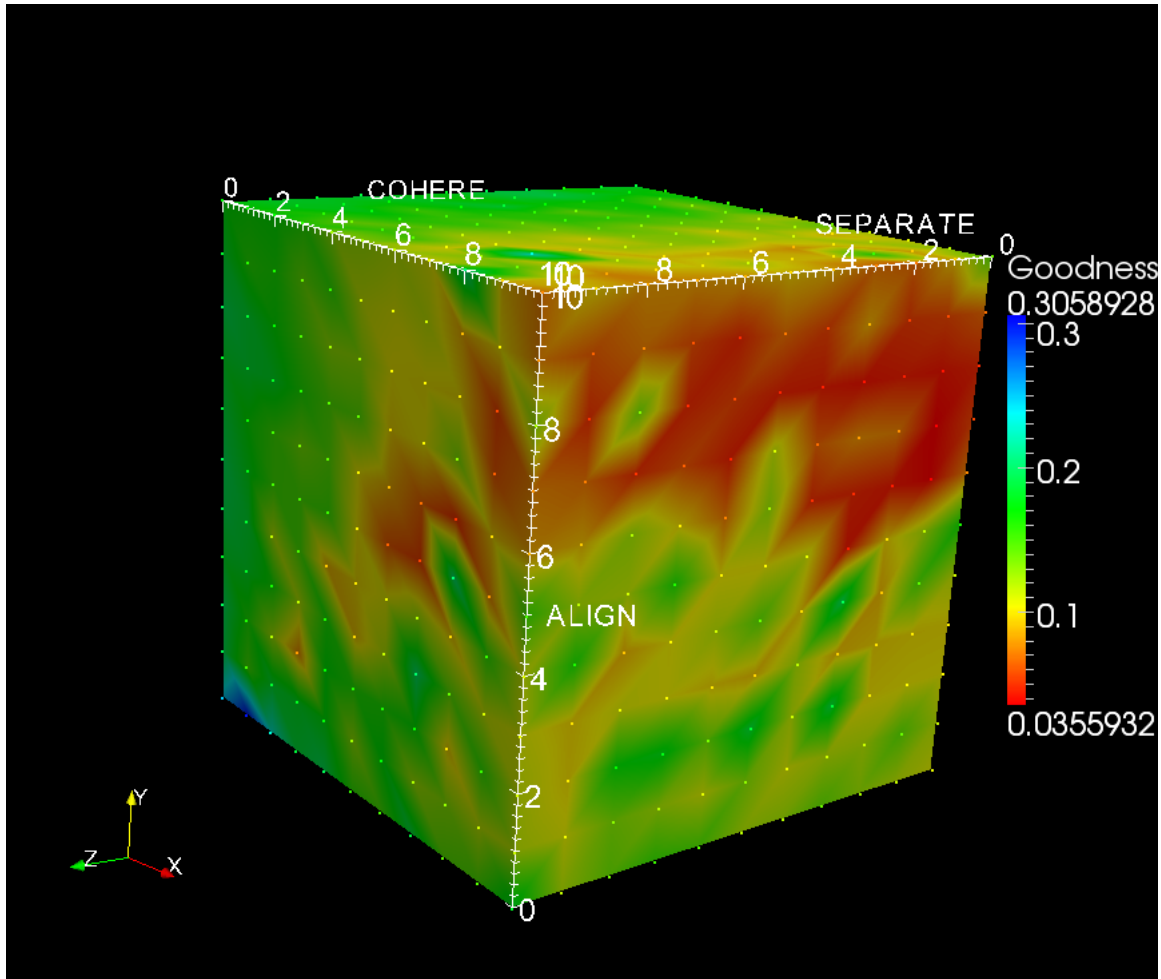


Figure 2. Search space as interpolated surface cube.

Using the same test harness, Behavior Space, each parameter search technique was run ten times and the outputs saved to a file. The outputs included the time it took to run each search, the parameters to the flocking run, the average goodness function value, and the number of “gen-eval-select” steps. Using this data, and the code written, the number of evaluations (200-timestep flocking runs) is calculated.

Results	Brute Force	Bracketing	Genetic	Steepest Descent
Time (Min)	~150	8.79850	2.06589	2.93913
Reliability	100%	70%	90%	100%
Goodness	0.03559	0.05466	0.08721	0.06869
Evaluations	1331	32	8.0	28.8

Table 2. Comparison of parameter search techniques.

- Time – The average time in minutes it takes for the search to come to a verdict.
- Reliability – The percentage of successful runs (runs having a goodness function under 0.1 in at least 30 iterations of the search).
- Goodness – The output flock goodness of each successful run of the search technique. For the bracketing, genetic and steepest descent, this is an average of the 10 runs output.
- Evaluations – The average number of 200 iteration flocking tests that are run (in successful runs).

5.0 Conclusions

The comparison of parameter search techniques shows that steepest descent is the most overall useful search technique, but each has its strengths and weaknesses.

Steepest descent is most “reliable” (as defined above) most likely because the parameter search space appears to be devoid of local minima and has a broad gradient for steepest descent to follow. It also has very good performance.

Bracketing was the least reliable, but the average goodness value of its output parameter combination is the closest to brute-force output. Its performance is worse than steepest descent and genetic, but still much better than brute-force. Bracketing’s reliability appeared to be impacted negatively by inadequate sampling of the search space due to the position of the gradient.

Although the genetic search technique is intriguing and its performance was better than the other three techniques, its average goodness value of its output parameter combination was the worst of the four. Since this technique was the only one that incorporated randomness, that may have affected its output goodness negatively.

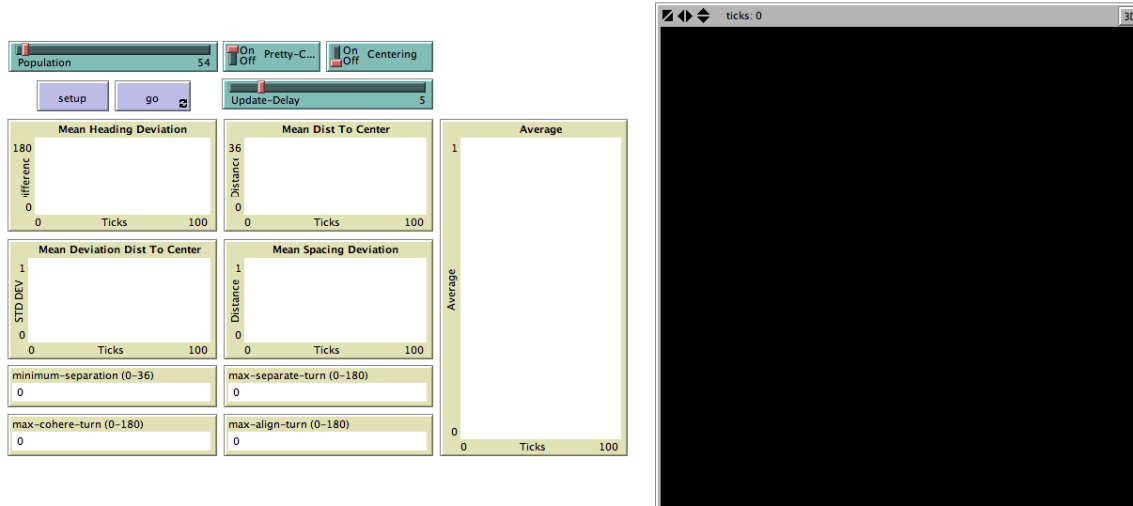
6.0 Significant Original Achievement

The most significant original achievement that was made by Team 70 was understanding the parameter search space. This is important to all of those interested in flocking, as the search space is very important to the search techniques run on it. Furthermore in order to understand the search space, the team made original contributions in equations to evaluate the flock.

7.0 Work Products

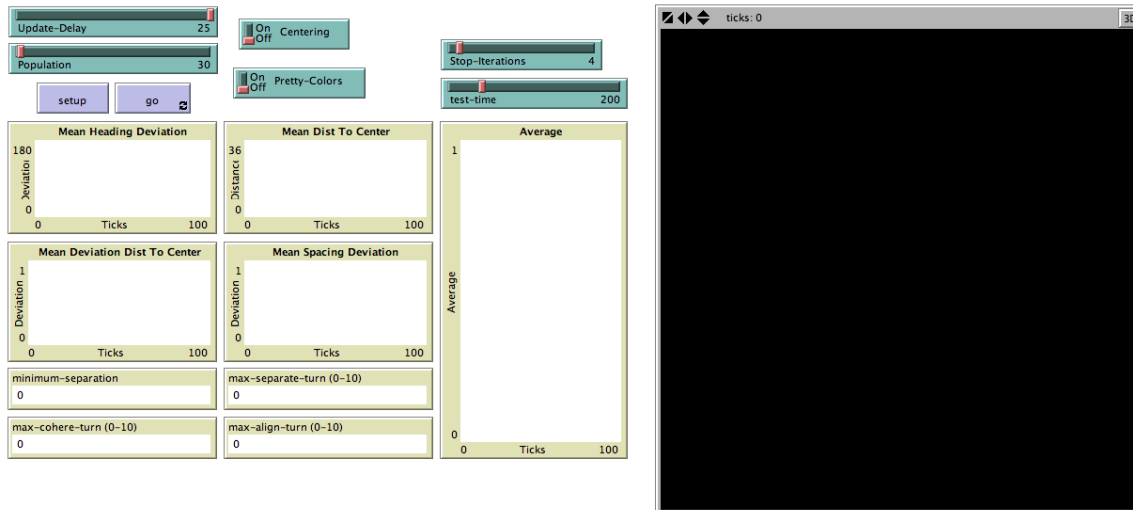
The code for the Behavior Space was stored with the NetLogo model. Each search technique has a different code base, but the goodness functions are identical. Code from original model is marked. BehaviorSpace code could not be included as it is stored in a GUI.

7.1 Flocking with Goodness Functions



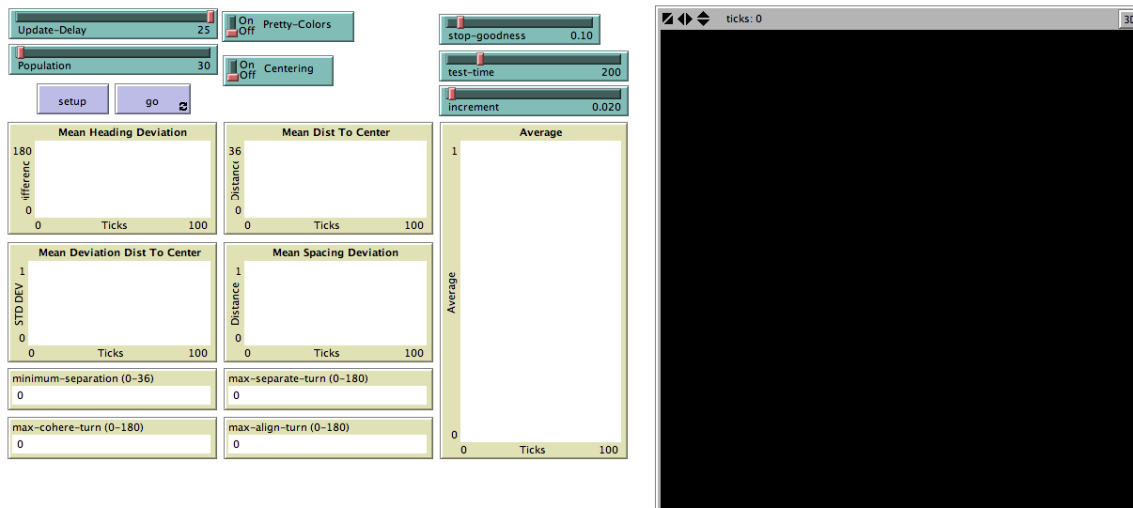
Code removed, see <http://www.challenge.nm.org/archive/09-10/finalreports/70.pdf> for the full report.

7.2 Bracketing



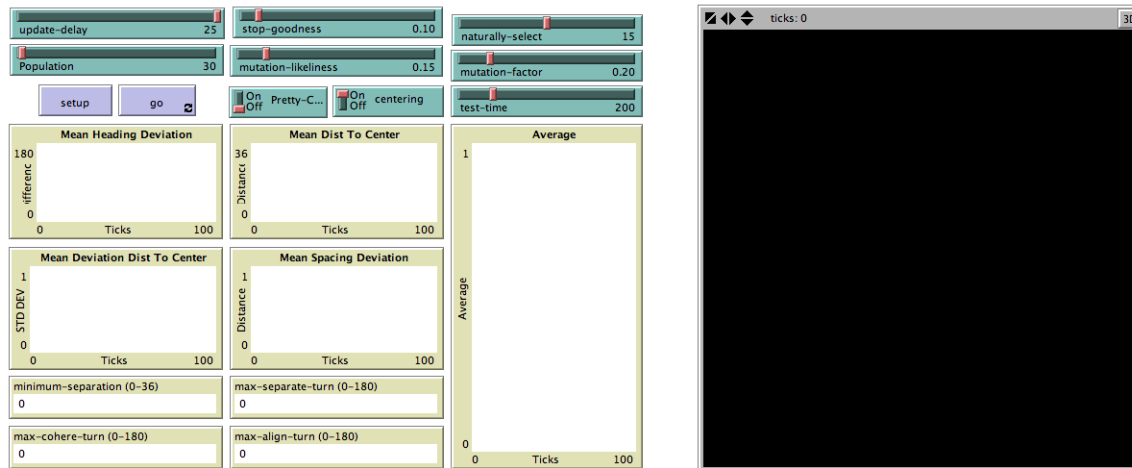
Code removed, see <http://www.challenge.nm.org/archive/09-10/finalreports/70.pdf> for the full report.

7.3 Steepest Descent



Code removed, see <http://www.challenge.nm.org/archive/09-10/finalreports/70.pdf> for the full report.

7.4 Genetic



Code removed, see <http://www.challenge.nm.org/archive/09-10/finalreports/70.pdf> for the full report.

8.0 Bibliography

- [1] Stephanie Forrest: Genetic Algorithms: Principles of Natural Selection Applied to Computation, 2007
- [2] Isaac Council, Lee Giles : Random Search For Multiple Layer Perceptron 2005
- [3] István Maros : Computational Techniques of the *Simplex Method* (International Series in Operations Research & Management Science) 2004
- [4] R. Fletcher: Practical Methods of Optimization 2005
- [5] Nigel Gilbert: Agent-Based Models in NetLogo (Quantitative Methods in Science) 2008
- [6] Nick Bennett, Bob Robey, and Tom Robey :Computational Solution Techniques In Mathematical Programming
- [7] Linge Bai and David Breen: Center of Mass in an Unbounded 2D Environment
- [8] Wikipedia – genetic algorithms and steepest descent.
- [9] Bob Robey, Tom Robey: Talk on optimization algorithms at Supercomputing Challenge Kickoff 2009.
- [10] The NetLogo flocking model:
Wilensky, U. (1998). NetLogo Flocking model.

<http://ccl.northwestern.edu/netlogo/models/Flocking>.

Center for Connected Learning and Computer-Based Modeling,
Northwestern University, Evanston, IL.

[11] ParaView: www.paraview.org.

9.0 Acknowledgements

First and foremost, we would like to thank the people of the Supercomputing Challenge. This project has opened the eyes of everybody on this team about the depths and usefulness of computer programming. We would like to thank in particular Bob Robey for helping us formulate an idea for a project.

Secondly, team 70 would like to thank Mr. Goodwin, our sponsor teacher, as he has encouraged us through this difficult journey and kept us enthusiastic about the task at hand. He has provided a warm and comforting atmosphere for our team, and the school, to get together and move forward in our work.

Thirdly, we would all like to thank our sponsors, Christine and Jim Ahrens for their expertise in computing and data visualization.

Fourthly, we would like to thank all of the judges. The judges took time out of their busy lives to listen to our proposals and interim reports and give us feedback on what we need to work on. They gave us positive feedback as well as suggestions for the future to make our project the best that it could be.

And last but not least, we would all like to thank our families, our moms, dads, sisters, and brothers.

The Metropolis Algorithm and Nanometer-Scale Pattern Formation

New Mexico
Supercomputing Challenge
Final Report
April 7, 2010

Team 5
Albuquerque Academy

Team Members:

Michael Wang
Jack Ingalls

Teacher:

Jim Mims

Project Mentor:

David Dunlap, Ph.D

Table of Contents

Executive Summary	Page 2
A Brief Summary of the Past Project	Page 3
Mathematical Model	Page 5
Boltzmann's Distribution	Page 7
Simple Two Energy System	Page 7
N Energy System	Page 8
The Metropolis Algorithm	Page 10
Algorithm	Page 10
Entropy	Page 10
Verifying the Metropolis Algorithm for a Two Energy System	Page 11
Energies Used in Our System	Page 12
Architecture of the Program	Page 14
Results and Discussion	Page 15
Future Plans	Page 17
Acknowledgements	Page 19
References	Page 20
Appendices	Page 21
Appendix A: Parallel Metropolis Algorithm	Page 21
Appendix B: Screenshots of program	Page 22

Executive Summary

When chemicals are layered on a surface, they begin to form patterns in order to reduce energy. This phenomenon is known as nanometer-scale pattern formation. This phenomenon plays a huge role in nanotechnology. If understood completely, it can be used to create devices at the nanometer scale. In addition, it would allow for cheap mass production.

In our previous project, we wrote a program that could simulate these patterns. However, we ran into several problems. We had to solve a set of differential equations that described the pattern formation process. Solving the equations required intense calculations, which slowed down our program significantly. In many cases, simulations had to be done overnight to obtain any useful results. In addition, we noticed that somehow mass was not conserved. We believe that the main culprit is numeric error.

In light of these issues, we seek to find a new way to simulate this phenomenon. In this project, we present a Monte Carlo method known as the Metropolis Algorithm that has successfully simulated the patterning phenomenon. This method of solving the problem provides us, and hopefully future users, with a short simulation time and a great amount of flexibility to allow us to study systems under a wide variety of conditions. In the previous project, extending the project to include many conditions might prove to be impossible, as one might not be able to derive the equations. On the other hand, with this project, including many conditions doesn't require more than a simple change in the code, thus making this program far more flexible.

A Brief Summary of the Past Project

When deposited on a solid surface (substrate), some chemicals rearrange to form patterns. The main factor that drives this pattern formation is free energy. In order to reach any sort of equilibrium, a system will try to minimize its total free energy. In the case of pattern formation, the system will separate into multiple phases, that is, multiple regions of different concentration. Each concentration corresponds to a minimum (a trough) in the free energy function as shown in the following figures (Suo and Lu, Forces that drive nanoscale self-assembly on solid surfaces, 2000).

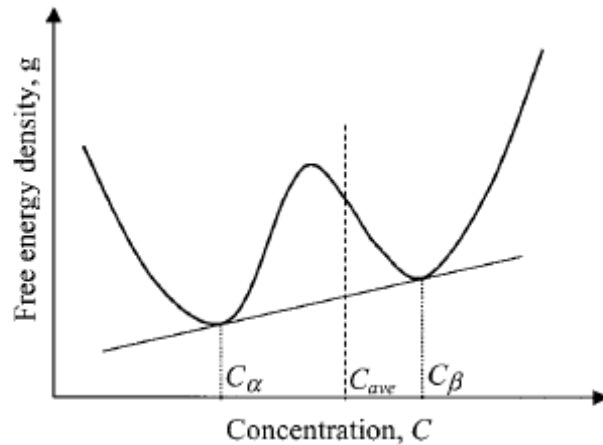


Figure 1: This is the free energy as a function of concentration. C_α and C_β correspond to the phases (below) α and β , respectively.

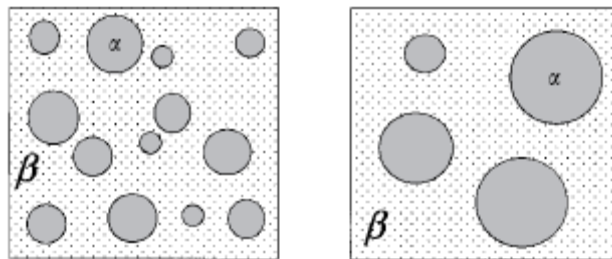


Figure 2: Regions α and β are the two phases the system separates into.

There are actually two other factors: surface strain energy and interface energy. These factors however, do not affect the separation process. Instead, they are involved in size selection. Surface strain energy is the energy due to an elastic deformation in the substrate, which occurs when there is a heterogeneous pattern. This is in some way analogous to having blocks of different masses resting on a bed of springs. The heavier blocks will tend to compress the strings more than the lighter ones. So, different concentrations will have different effects on the substrate. Interface energy, also known as phase boundary energy, is the energy associated with a difference in concentration, or the chemical gradient. It is like saying that a particle with zero neighbors will have an energy (likely higher) different from that of a particle with more than zero neighbors. So, the longer a boundary is, the more interface energy there will be.

Now, how do these affect size selection? It should be clear that interface energy increases with the sum of the lengths of all the boundaries. Thus the system will try to reduce the total boundary length. It turns out that the way to do this is by clumping like regions together. In Figure 2, we see that the smaller regions group together into larger regions. Will this continue until all that is left is one large region? The answer would be yes if surface strain energy were not included. As the size of each region increases, the larger the “heavier blocks” become and the more deformed the substrate becomes. To reduce the strain energy, the size of the regions must decrease. As you can probably guess, these two factors will eventually balance and the system will settle down in equilibrium.

Mathematical Model

A diffusion equation has been developed by Lu and Kim to describe the process above.

$$\frac{\partial C}{\partial t} = \nabla^2 \left(\ln \frac{C}{1-C} + \Omega(1-2C) - 2\nabla^2 C - \frac{Q}{\pi} \iint \frac{(x_1 - \xi_1) \frac{\partial C}{\partial \xi_1} + (x_2 - \xi_2) \frac{\partial C}{\partial \xi_2}}{\left((x_1 - \xi_1)^2 + (x_2 - \xi_2)^2 \right)^{\frac{3}{2}}} d\xi_1 d\xi_2 \right)$$

This equation is solved by using the Fourier Transform and a semi-implicit difference method. The equation simplifies greatly to $\frac{\partial C}{\partial t} = -k^2 \hat{P} - 2(k^4 - k^3 Q) \hat{C}$, where

\hat{P} is the Fourier Transform of $\ln \frac{C}{1-C} + \Omega(1-2C)$. Applying the semi-implicit method,

we end up with $\hat{C}_{n+1} = \frac{\hat{C}_n - k^2 \hat{P}_n \Delta t}{1 + 2(k^4 - k^3 Q) \Delta t}$. Because we can't intuitively "see" Fourier

space, we must somehow transform this equation back into real space. This is done with the Fast Fourier Transform (FFT). Also, there are no known transforms for P .

Therefore, we must calculate P in real space and then transform it to Fourier space.

This procedure produces interesting patterns as well as helping us understand the mysterious nanoworld. However, there are two issues that we must address. The first issue deals with time. This procedure is very computationally intensive due to calling the FFT multiple times every time step. In order to get interesting and useful results, we sometimes are required to run the simulation for several hours. Though this isn't an especially long period of time, it can still pose problems.

The second issue deals with stability. It is clear from the numeric methods that errors can build up over time. We have noticed that mass is not conserved. Mass seems to appear out of nowhere. We believe that there are two possible causes for this: numeric

error or just something that wasn't taken into account in the equations. Whatever the cause, the simulations cannot give any useful results if mass is not conserved.

In light of these issues, we sought another simulation method, in particular, a Monte Carlo method. Professor David Dunlap suggested to us the Metropolis-Hastings Sampling Algorithm.

Boltzmann's Distribution

The Boltzmann's distribution describes the probability that a system is in some particular state. For example, Boltzmann's distribution is frequently used to describe the distribution of velocities of particles in a gas (also known as the Maxwell-Boltzmann's distribution). In a more general case (not just kinetic energy), the Boltzmann's distribution either describes the probability that a particles has a specific energy or roughly how many particles have one specific energy. In this section, we show some brief simplified derivations of Boltzmann's distribution in a two energy system and a n energy system.

Deriving Boltzmann's Distribution for a Simple Two Energy System

Let E_1 and E_2 be the two possible energies a particle can have. Suppose our system is made up of N particles and M lattice sites on a grid. M_1 of the sites have energy E_1 while M_2 have energy E_2 . We want to determine how many particles have energy E_1 and how many, energy E_2 . We assume that there are N_1 and N_2 particles having energies E_1 and E_2 , respectively. To find N_1 and N_2 , will minimize the free energy of the system. The free energy F is defined as $E - kT \ln W$, where E is the energy, k is Boltzmann's constant ($1.3806505 \times 10^{-23} J/K$), T is the temperature, and W is the number of possible microstates or configurations. The number of microstates is the number of ways N_1 particles can be placed into M_1 locations times the number of ways N_2 particles can be placed into M_2 locations, or $W = \binom{M_1}{N_1} \binom{M_2}{N_2}$. Substituting this in with

$$E = E_1 N_1 + E_2 N_2, \text{ we have } F = E_1 N_1 + E_2 N_2 - kT \ln \left(\frac{M_1!}{N_1! (M_1 - N_1)!} \frac{M_2!}{N_2! (M_2 - N_2)!} \right).$$

We now wish to minimize F .

Rewriting using logarithm rules, we obtain

$$F = E_1 N_1 + E_2 N_2 - kT (\ln(M_1!) - (\ln(N_1!) + \ln((M_1 - N_1)!)) + \ln(M_2!) - (\ln(N_2!) + \ln((M_2 - N_2)!)))$$

Using Stirling's approximation, $\ln(K!) \approx K \ln K - K$ for large K ,

$$F = E_1 N_1 + E_2 N_2 - kT (M_1 \ln M_1 - M_1 - (N_1 \ln N_1 - N_1 + (M_1 - N_1) \ln(M_1 - N_1) - (M_1 - N_1))) + M_2 \ln M_2 - M_2 - (N_2 \ln N_2 - N_2 + (M_2 - N_2) \ln(M_2 - N_2) - (M_2 - N_2)))$$

Now we can minimize F using $N_2 = N - N_1$.

$$\frac{dF}{dN_1} = E_1 - E_2 - kT (\ln(M_1 - N_1) - \ln N_1 - \ln(M_2 - N_2) + \ln N_2), \text{ or}$$

$$\frac{dF}{dN_1} = E_1 - E_2 - kT \ln \frac{(M_1 - N_1) N_2}{(M_2 - N_2) N_1} = 0.$$

Knowing that $N = N_1 + N_2$, we can readily solve for N_1 and N_2 .

Deriving Boltzmann's Distribution for a n Energy System

This is almost identical to the two energy system, only now we have n different energies. F now is defined as $F = \sum_{i=0}^n E_i N_i - kT \ln \prod_{i=0}^n \binom{M_i}{N_i}$, or equivalently

$$F = \sum_{i=0}^n E_i N_i - kT \sum_{i=0}^n \ln \frac{M_i!}{N_i! (M_i - N_i)!}.$$

Simplifying and using Stirling's approximation, we have

$$F = \sum_{i=0}^n E_i N_i - kT \sum_{i=0}^n (M_i \ln M_i - M_i - (N_i \ln N_i - N_i + (M_i - N_i) \ln(M_i - N_i) - (M_i - N_i))).$$

Using Lagrange multipliers with the constraint $\sum N_i = N$, we find that $\frac{\partial F}{\partial N_i} = \frac{\partial F}{\partial N_j}$ for all

i and j . Therefore, knowing $\frac{\partial F}{\partial N_i} = E_i - kT \ln \frac{M_i - N_i}{N_i}$, we obtain

$$E_i - kT \ln \frac{M_i - N_i}{N_i} = E_j - kT \ln \frac{M_j - N_j}{N_j}.$$

Assuming the system is “large” (i.e. $M_i \gg N_i$) and using the same approximation, we

have $N_j = \frac{M_j}{M_i} N_i e^{\frac{E_i - E_j}{kT}}$. Summing over all j , we get $\sum_{j=0}^n N_j = N = \sum_{j=0}^n \frac{M_j}{M_i} N_i e^{\frac{E_i - E_j}{kT}}$. So,

$N_i = \frac{N}{\sum_{j=0}^n \frac{M_j}{M_i} e^{\frac{E_i - E_j}{kT}}}$. However, if we do not assume that $M_i \gg N_i$, then the equation

becomes significantly harder to solve. We will not show it here.

Metropolis Algorithm

Boltzmann's distribution can be useful when figuring how particles are distributed. Other examples include the distribution of velocity of gas particles (just let E be kinetic energy). Analytically, the above process is how we would derive the distribution. How would we do it computationally? This is where the Metropolis Algorithm comes in (there are other ways). In the Metropolis Algorithm, we essentially check whether or not a particle will move based on the energy change of that move. The rules are incredibly simple.

1. If the energy change is negative, accept the move.
2. If the energy change is positive, generate a random number between 0 and 1. If that random number is less than $e^{-\frac{\text{energy change}}{kT}}$, accept the move. Otherwise, reject it.

Entropy in the Metropolis Algorithm

The Metropolis algorithm at first sight seems to not include entropy, the randomness of a system. After all, it is based solely on energy reduction. The entropy, however, is actually really subtle. There are two ways to think it. The first way is to examine how particles are moved in the algorithm. The direction of movement is random. In addition, it is clear that a particle can only move to an unoccupied location. Therefore, if that particle is in an organized group, the only direction it can move is away from that group, thus slightly disturbing the order in the system. The second way to understand entropy is to examine the energies. In many cases, the energies themselves secretly encode entropy. In our case, that energy is the surface strain energy. If particles begin clumping together, the strain induced on the surface increases, which in turn increases the energy. That increase in energy due to a high concentration of particles in

one region causes those particles to spread out, once again disturbing some sort of order. So, entropy is in fact included in the algorithm.

Verifying the Metropolis Algorithm

To determine whether we coded the algorithm rules correctly, we tested our code on one distribution. That distribution is based on Boltzmann's distribution in a two energy system. For this simulation, the simulated region is a box having side length 200, meaning that there are a total of 40000 ($=M$) sites on the lattice. This box is divided in to two rectangles, one with 10000 ($=M_1$) sites and the other with 30000 ($=M_2$) sites. The rectangle containing 10000 sites is assigned a unit-less energy of 3 while the other rectangle is assigned an energy of 0 (we let Boltzmann's constant k and T be 1 for simplicity). In our simulations, we placed down 20000 particles.

To test our code, we look at the ratio of the number of particles in one region to the number of particles in the other region. From our derivation of Boltzmann's distribution for a two energy system, we find that $\frac{N_2}{N_1} = 0.0423$, where N_2 is the number of particles having energy 1 and N_1 is the number of particles having energy 0.

<u>Number of moves</u>	<u>Ratio N_2/N_1</u>
100000	0.33129202
1000000	0.32231404
2000000	0.31648235
10000000	0.28924128
20000000	0.27048660
100000000	0.20141767
200000000	0.15667110
300000000	0.12549240
400000000	0.10674561
500000000	0.09481060
600000000	0.08483402
700000000	0.07874865
1000000000	0.05982725

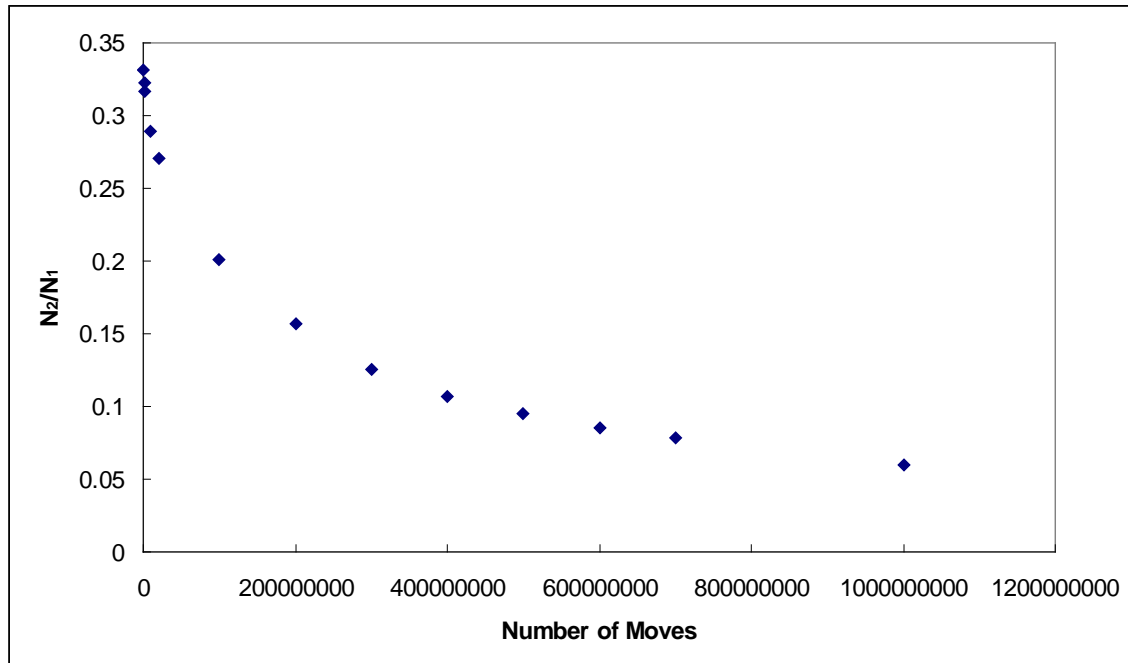


Figure 3: Graph of N₂/N₁ as a function on the number of moves. The curve formed by the points approach ~0.04-0.05

We did not run the simulation long enough to obtain points around 0.42. However, based on the table and graph, it is clear that 20000000000 moves, the ratio should begin to hover around 0.42.

Energies in Our System

There are two energies defined in our system: interaction energy and the misfit strain energy. The interaction energy is like the boundary energy in our previous project while the misfit strain energy is related to the surface strain energy in our previous project.

The interaction energy essentially models bonding and is determined by the number of neighbors. In our specific system, the maximum number of neighbors a particle can have is four. In our program, the energy is a function of the number of neighbors. In many cases, as that number increases, the energy decreases. If a particle

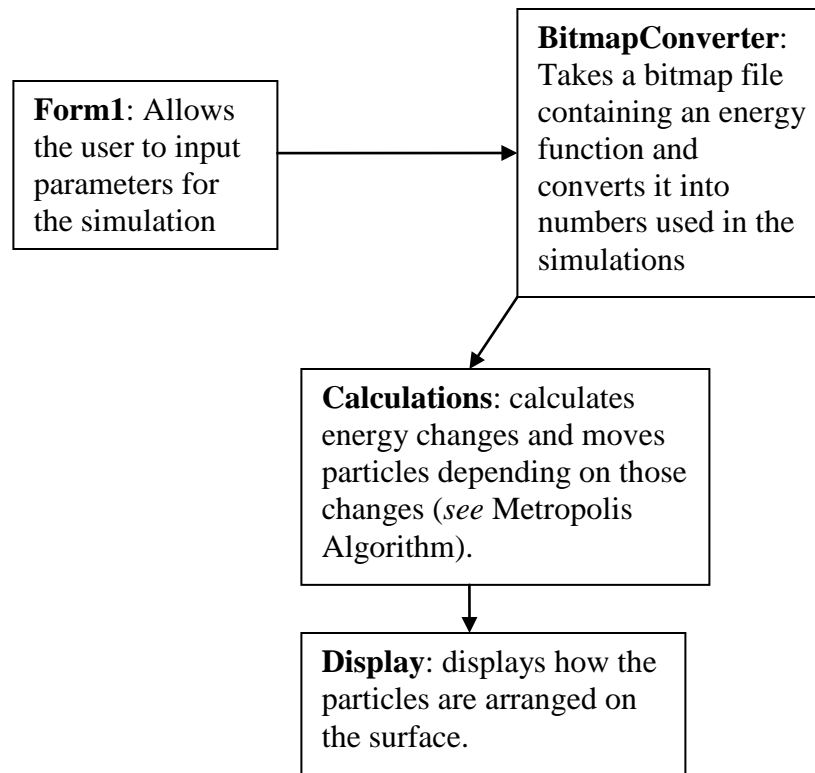
loves to “bond” or interact with other particles, then its energy will drop significantly for each addition particle it finds.

The misfit strain energy essentially models how much strain a layer of particles induces on the surface. The lattices constants for the particles and the surface are usually different, which creates misfit. Particles typically want to align themselves with the surface to reduce energy. However, due to the size difference, many particles will not be able to align and will instead become offset. This creates strain and potential energy in the surface. Below is a side view of the particles and surface.



To calculate the misfit energy, we look at the distances between the midpoints of the film and the lattice. One can view each line above as one particle. In our program, we select two adjacent particles in the film. That gives us an interval between those two particles. We then find all of the midpoints that lie on that interval. In most cases, there is only one such midpoint. Once we find all of the corresponding midpoints, we take the average of the distances between the midpoint of the two particles and the corresponding midpoints in the lattice/surface. A particle that is aligned with the surface will have an energy of 0 because the midpoints will line up. However, if the particle is offset by a small amount, the midpoints will not line up, thus generating energy. This energy we describe is a function of the distance between the midpoints. For simplicity and flexibility, we let that function be a polynomial. Users can include as many terms in the polynomial as they wish, depending on the shape of the function.

Architecture of the Program



The structure of the program is straightforward. The program is written in C#, which is very similar to Java. In Form1, the user first inputs values for the parameters and starts the simulation. The calculations are performed in Calculations and then the results are displayed in Display. The calculations can be run over and over until satisfactory results are obtained.

The BitmapConverter reads the hexadecimal numbers contained in the bitmap file and assigns energy levels to those numbers, or colors if one looks at the picture (e.g. black = 1 and white = 0).

Results and Discussion

The three most important results that we must reproduce with our program are quantum dots (patches of particles), serpentine stripes (particles arranged in snake-like shapes), and quantum pits (patches or “holes” where there are no particles). At lower concentrations, we should be getting quantum dots. As we increase the concentration or the number of particles, the results should transition from quantum dots to serpentine stripes and finally from serpentine stripes to quantum pits. Physically, this makes sense. When the number of particles is low, serpentine stripes cannot form because the energy can still be lowered by breaking the serpentine stripes into quantum dots. As the number of particles increases, it becomes difficult to form quantum dots because the limited room would create rather large patches, which actually would have a higher energy than serpentine stripes. Once the particles cover a majority of the surface, quantum dots become impractical (they would be enormous). There would not be enough room to form serpentine stripes. They would be so close together that they would begin to merge. Thus, quantum pits form.

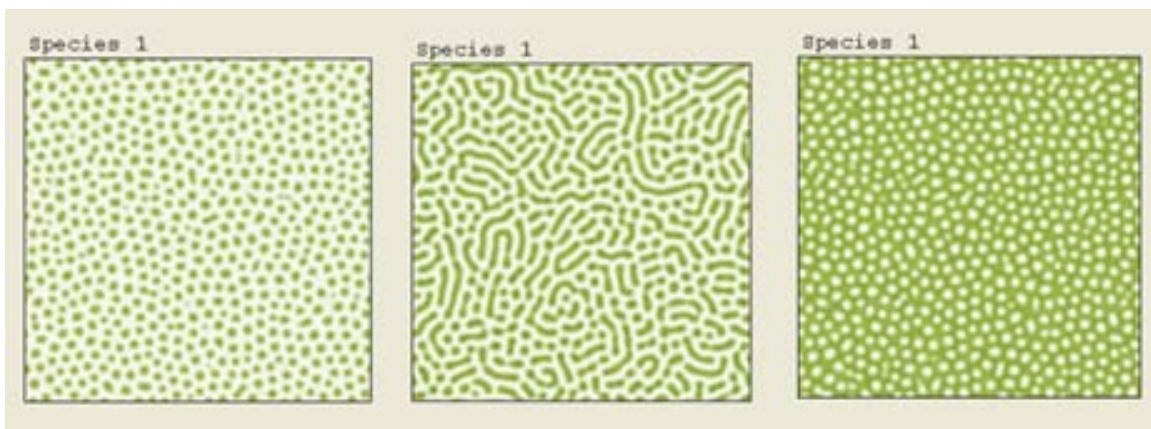


Figure 4: Simulations from the past project. In the image on the far left, we see quantum dots (low concentration). In the middle image, we see serpentine stripes (medium concentration). In the image on the far right, we see quantum pits (high concentration).

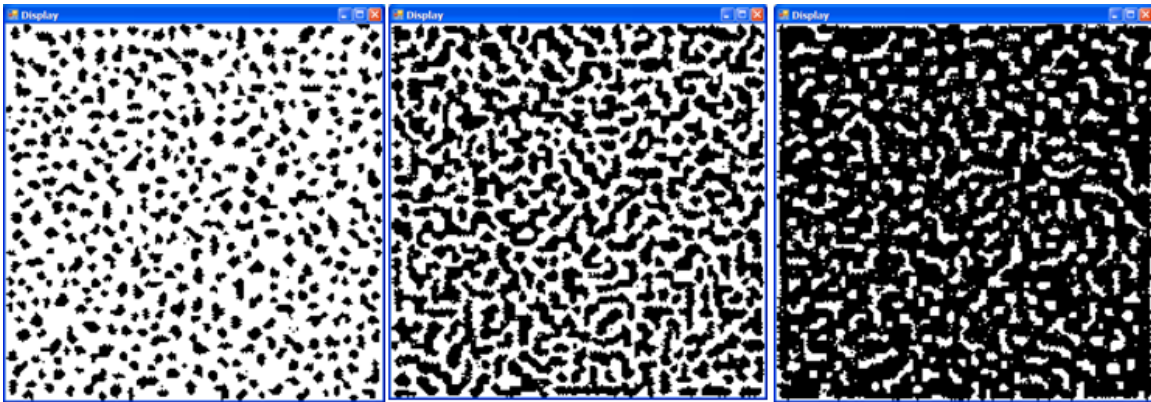


Figure 5: As in Figure 4, we see the transition from quantum dots to serpentine stripes and from serpentine stripes to quantum pits. Note that these simulations are more microscopic than those in Figure 4.

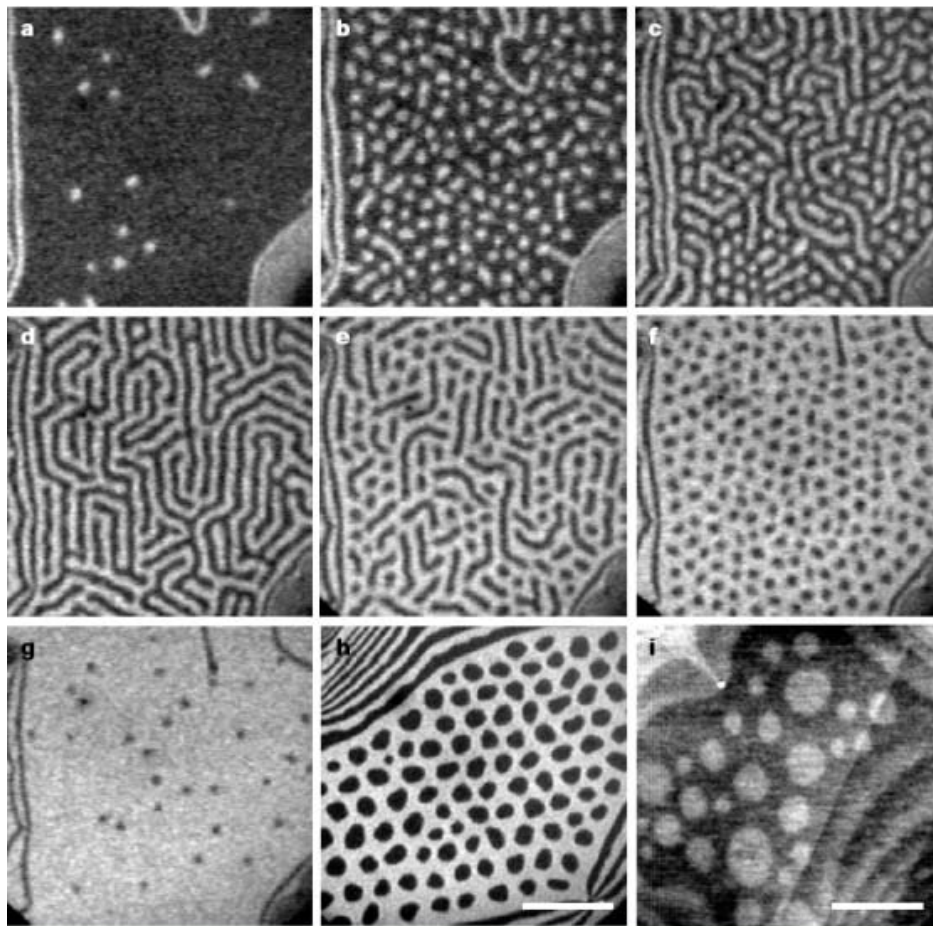


Figure 6: Experimental observations of the nanoscale self-assembly of Pb (lead) on Cu (copper) (Plass et al., 2001). A transition from quantum dots through serpentine stripes to quantum pits can be clearly seen in b-f.

The final simulation we perform shows the effect of misfit strain energy. As predicted, misfit strain energy (also known as surface strain energy) should cause the system to form finer patterns. For example, quantum dots should be smaller. Below, we compare a simulation using small, almost negligible strain to one that uses a much larger strain.

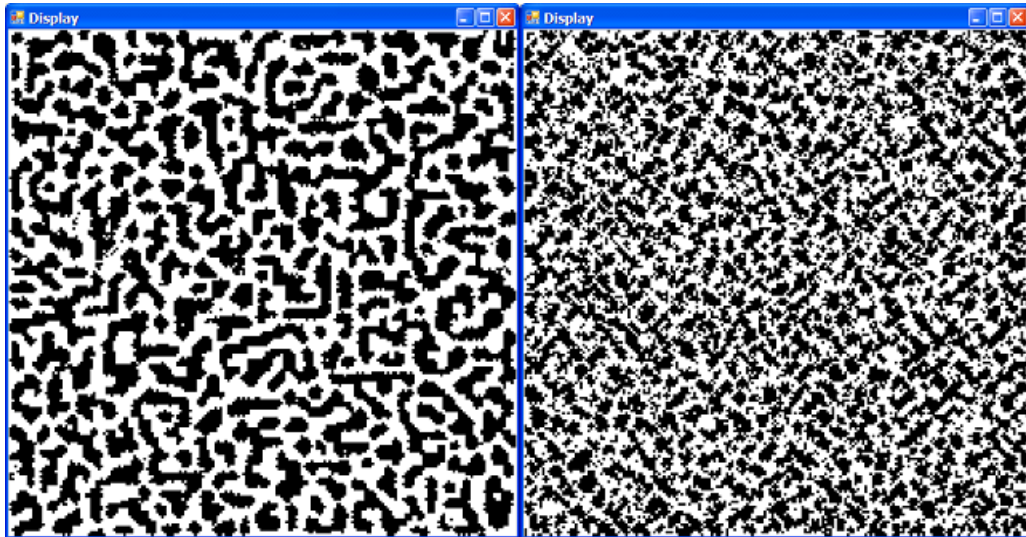


Figure 7: This comparison shows the effect of strain energy. The figure on the right didn't use any strain energy while the figure on the left used $10x+10x^2$ (x is the distance between midpoints) as the strain energy function, which is significantly larger. Due to the strain energy, the patterns in the system on the left become much finer because the system is more sensitive to concentration.

Based on the simulations above and the numerous reruns, our program has successfully simulated the qualitative features of our system. We successfully reproduced quantum dots, serpentine stripes, and quantum pits. These results match accurately with our old program and experimental results. In addition, we successfully included the effect of strain on the patterns.

Future Plans

There were many things that we could not include in our finished product. We finished the most essential parts of the program. There are actually three improvements

that we plan on making. The first has to do with parallel computing. The Metropolis algorithm can be run on multiple cores. Not surprisingly, the new algorithm is called the Parallel Metropolis algorithm (*see* Appendix A for a brief description). We would also like to expand our system more. In other words, generalize it. Now that we have the basic components finished, we can start adding more energy so that we can apply our program to many other systems. For example, we can study how dipoles might arrange on the surface or how an electric or magnetic field might effect pattern formation, which could be useful to know if nanoscale circuits are used. All of these additions are extremely easy to include. Unlike the past project, we will not have to derive, if possible, any differential equations when we make the system more complicated. It might be impossible to derive equations for such complicated systems. Instead, we simply determine what new energies to include. Finally, we would like to include heterogeneous initial distributions, that is, the user can put whatever initial distribution (e.g. a circuit) and see how it changes under certain conditions.

Acknowledgements

We would like to thank our mentor, Professor David Dunlap, for his invaluable help in demystifying the Metropolis Algorithm and giving us a brief introduction to thermodynamics. We would also like to tank Jim Mims for supporting us along the way.

References

- Chen, L.-Q. and Shen, J., Applications of semi-implicit Fourier spectral method to phase field equations, *Comp. Phys. Commun.* 108 (1998) 147–158.
- Hu, Shaowen, Girish Nathan, Fazle Hussain, Donald J. Kouri, Pradeep Sharma, and Gemunu H. Gunaratne. "On Stability of Self-Assembled Nanoscale Patterns." *Journal of the Mechanics and Physics of Solids* 55 (2007): 1357-1384. [ScienceDirect](http://www.sciencedirect.com/). Elsevier. 28 Nov. 2007 <<http://www.sciencedirect.com/>>.
- Johnson, K. L. "Point Loading of an Elastic Half-Space." *Contact Mechanics*. Cambridge, U.K.: Cambridge University Press, 1985. 45-83.
- Kaganovskii, Yu. S., L. N. Paritskaya, and V. V. Bogdanov. "Kinetics and Mechanisms of Intermetallic Growth By Surface Interdiffusion." *Mat. Res. Soc. Symp. Proc.* 527 (1998): 303-307.
- Lu, Wei. "Theory and Simulation of Nanoscale Self-Assembly on Substrates." *Journal of Computational and Theoretical Nanoscience* 3.3 (2006): 342-361.
- Lu, Wei, and Dongchoul Kim. "Dynamics of Nanoscale Self-Assembly of Ternary Epilayers." *Microelectronic Engineering* 75 (2004): 78-84. [ScienceDirect](http://www.sciencedirect.com/). 26 Feb. 2004. Elsevier. 28 Nov. 2007 <<http://www.sciencedirect.com/>>.
- -. "Patterning nanoscale Structures by Surface Chemistry." *Nano Letters* 4.2 (2004): 313-316.
- -. "Simulation on Nanoscale Self-Assembly of Ternary-Epilayers." *Computational Materials Science* 32 (2005): 20-30. [ScienceDirect](http://www.sciencedirect.com/). Elsevier. 8 Dec. 2007 <<http://www.sciencedirect.com/>>.
- Plass, Richard, Julie A. Last, N. C. Bartelt, and G. L. Kellogg. "Self-Assembled Domain Patterns." *Nature* 412 (Aug. 2001): 875. 25 Mar. 2008 <<http://www.nature.com/>>.
- Ratsch, C. and Venables J. A., Nucleation Theory and the early stages of thin film growth, *J. Vac. Sci. Technol. A*, Vol. 21, No. 5 (2003) S96-S109
- Roduner, Emil. *Nanosopic Material: Size-Dependent Phenomena*. Cambridge: The Royal Society of Chemistry, 2006.
- “Metropolis Algorithm Statistical System and Simulated Annealing.” N.d. *Physics 170*. Web. 7 Apr. 2010. <<http://kossi.physics.hmc.edu/courses/p170/Metropolis.pdf>>.

Appendix A: Brief Description of the Parallel Metropolis Algorithm

As its name suggests, the Parallel Metropolis algorithm is a parallel version of the Metropolis algorithm. To run the parallel version, the simulation grid is divided into sections, each sections is handled by one processor.

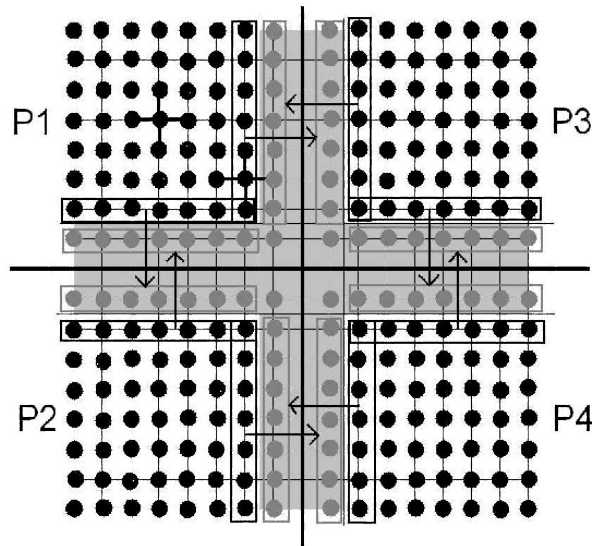
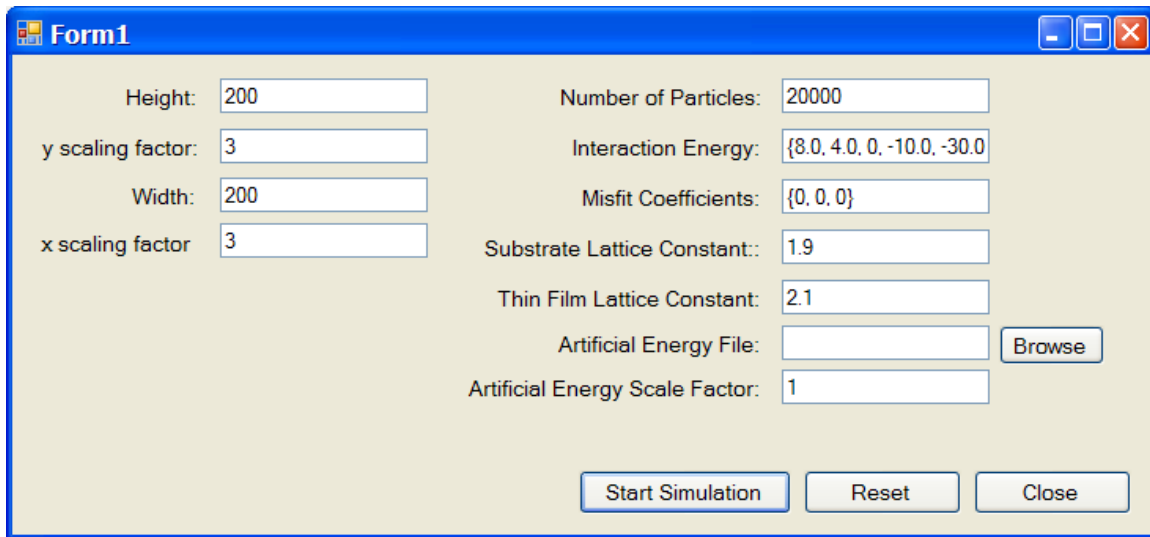


Figure 8: The schematics for the Parallel Metropolis Algorithm. Gray cells represent ghost cells.
<http://www.fysik.uu.se/cmt/berg/node31.html>

In Figure 8, the grid is divided into four quadrants. Each quadrant contains ghost cells (gray areas) that hold information about the adjacent quadrant. As the program runs, the ghost cells are continuously updated. Currently, our program runs significantly faster than the program in our past project. However, we noticed that as the system becomes larger (more than a 200x200 grid), the program gets slower and slower. By implementing the Parallel Metropolis Algorithm, we hope to further reduce the simulation time, which would allow us to simulate much larger and more interesting systems.

Appendix B: Screenshots of the Program



The screenshot shows a window titled "Form1" with a light beige background and a blue title bar. It contains several input fields and buttons. The parameters are as follows:

Parameter	Value
Height	200
Number of Particles	20000
y scaling factor	3
Interaction Energy	{8.0, 4.0, 0, -10.0, -30.0}
Width	200
Misfit Coefficients	{0, 0, 0}
x scaling factor	3
Substrate Lattice Constant	1.9
Thin Film Lattice Constant	2.1
Artificial Energy File	[Empty]
Artificial Energy Scale Factor	1

Buttons at the bottom include "Start Simulation", "Reset", and "Close". A "Browse" button is next to the "Artificial Energy File" field.

Figure 9: Form 1. Here, users can input parameter values. The x and y scaling factors just change the pixel size of each particle.

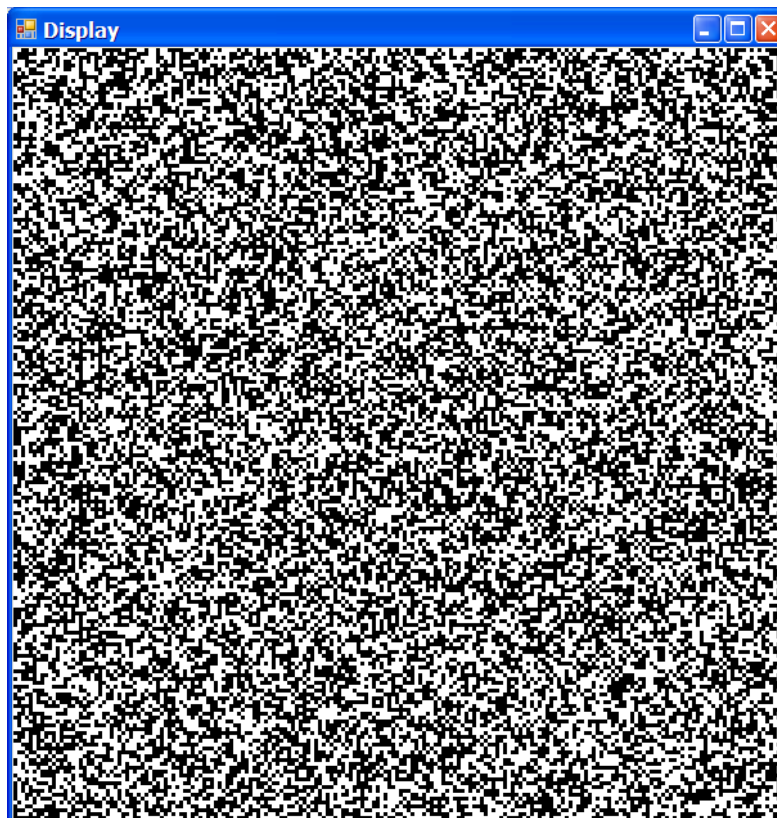


Figure 10: Display screen. This is where the result is displayed. The current image is one of a random initial distribution. Attached to this screen is a little control device on which the user can continue the calculations or exit the program.

Arbitrary Precision Integers on the Cell Processor

New Mexico

Supercomputing Challenge

Final Report

April 6, 2010

Team #36

Desert Academy

Team Members

Megan Belzner

Matt Rohr

Bjorn Swenson

Teacher / Project Mentor

Thomas Christie

THIS PAGE HAS BEEN LEFT INTENTIONALLY BLANK

Table Of Contents

0 Executive Summary	4
1 Introduction	5
1.1 Problem	5
2 Implementation	6
2.1 Single vs. Multi-core Operations	7
2.2 Data Type Development	7
2.3 Limited Success	8
2.4 Function List	9
2.5 Software Used	16
2.6 Literature Used	16
3 Results	17
3.1 Comparison to GMP	18
4 Analysis	20
4.1 Parallelizability / Comparison to GMP	21
5 Conclusion	23
5.1 Most Significant Original Achievement	24
6 Resources	24
Appendices	
Appendix A – Code	25
Appendix Z – EPIC FAIL	37

0 Executive Summary

The Cell processor is used in the Playstation 3 gaming console and the Roadrunner supercomputer at Los Alamos National Labs. It is a microprocessor specially designed to do calculations quickly with high data throughput. It has eight cores to process data and one core to manage the data flow, and provides many advantages, such as efficient calculation with the vector data type which allows multiple operations to be carried out in a single CPU cycle. In addition, all of the cores can be used at the same time and can communicate between each other.

Despite all this, there has never been any efficient use of the Cell for arbitrary precision calculations. Our goal in this project was to make an arbitrary precision integer library that utilized the special capabilities of the Cell to allow for extremely fast calculations on any size integer.

We programmed our library from scratch on a Playstation 3 using C. We created a new data type, called a vecthor. Each vecthor is an array of vectors that can hold up to [any multiple of] 1024 digits. We then proceeded to create functions for the standard operations commonly used in number theory and cryptography, namely addition, subtraction, multiplication, modding, and taking powers.

Throughout the creation of our library, we paid special attention to efficiency in memory usage and speed. We utilized many Cell-specific functions, and constantly tested our functions for speed. We collected data for multiple number lengths, and compared our results to the leading arbitrary precision library, GMP.

Our intention was to create a library optimized for the Cell processor that could be implemented separately and simultaneously on all of the cores and on multiple networked PS3s. We intended it to be used primarily for number theory applications such as prime finding as well as encryption and decryption.

However, even with meticulous code optimizations and the power of the Cell processor, our library failed to meet the mark. When compared to GMP, the results were dismal. GMP outperformed us by several orders of magnitude in most cases, even for relatively simple operations. We concluded that the power of the Cell processor is very hard to extract and manipulate. In general, the difficulty of programming and managing memory is more trouble than it is worth when dealing with foreign data types and machine instructions.

1 Introduction

In the year 2001, Sony, Toshiba and IBM (collectively called "STI") began work on the Cell Broadband Engine (CBE). The Cell is a microprocessor with a heterogeneous design: 8 cores processes data, and one core controls job execution and manages data flow. It was built around IBM's PowerPC technology, and is used in both Sony's Playstation 3 gaming system and IBM Supercomputers, including the Roadrunner at Los Alamos National Labs. All applications, from modeling physical simulations to multimedia, required that the processor have superior vector and floating point performance.

In its current design, the Cell has 9 cores - one PPU (Power Processing Unit) and eight SPUs (Synergistic Processing Unit), all of which run at 3.2GHz. The PPU is an ordinary PowerPC processor and serves to organize data flow and job execution between the other cores. The SPUs are specially designed processors theoretically capable of 25.6 GFLOPs each, and can perform operations on 128-bit data types called vectors. IBM and third parties have developed libraries on the Cell tailored for graphics and modeling, such as arbitrary precision floating point arithmetic and fast matrix multiplication. However, currently no arbitrary precision integer library exists which utilizes the Cell's unique capabilities, meaning the Cell is not being used for number theory or cryptography.

1.1 Problem

Our goal with this project was to create an arbitrary precision integer library for the Cell processor using the C programming language. The Cell processor on the Playstation 3 has a PPU (main processor) and 6 usable SPUs, since one is dedicated to the PS3's operating system and one is disabled by default. As mentioned above, each SPU can do basic arithmetic operations on 128-bit data types, or 4 integers at once. This means that, depending on the parallelizability of the algorithms, we thought we might be able to increase the speed of arithmetic operations on a single core up by up to 4x. If we could make each SPU operate at maximum capacity, each Cell could do 6 to 24 times as many arbitrary-precision operations as a single-core 3.2GHz processor in a given amount of time. Combined with the fact that MPI can be used to make multiple PS3s work on a problem in tandem, we thought that several networked PS3s could operate as a formidable machine for number-theoretical operations like prime finding and encoding/decoding information.

2 Implementation

We chose to work on the Cell because of the calculating speed of the 6 usable individual SPUs, the availability the Cell in the form of the Playstation 3, and the SIMD (Single Instruction Multiple Data) capabilities of the Cell Processor. The SPU's 128-bit vector data type can be divided in various ways, such as 4 integers, 8 shorts, 2 longs, etc. Included in the Cell's libraries provided by IBM are functions called intrinsics which map directly into assembly language instructions. One such intrinsic function allows multiple numbers to be added together in a single computation like so:

```
Vector 1:{230597, 17345, 5198357, 3567}
          +      +      +      +
Vector 2:{3298673, 839764, 235, 46082}
          =      =      =      =
Result: {3529720, 857109, 5198592, 49649}
```

This effectively means that 4 integers can be added simultaneously.

We created a new data type to take advantage of the speed of the Cell's intrinsic functions, which are SPU functions that map directly onto assembly language instructions. Our library of functions is called "thor," so we named our data type "vecthor." Each vecthor consists of an array of [some multiple of] 32 vectors, plus one more that contains the size of the vecthor. Each vector contains 4 integers of maximum 8 digits each, and the number is broken into 8-digit groups and inserted in the vectors. Of course, a signed integer can have a value up to $2^{31} = 2\ 147\ 483\ 648$, but we chose 99 999 99 as our "BIGGEST_INT" so that we could add several vectors together without the results overflowing the 2^{31} value.

In a vecthor, the first vector contains the metadata: the first integer (as well as the last one) is how many vectors are registered in memory as being part of the vecthor (not including the size vector and always a multiple of 32), the second integer contains the "virtual" size – how many vectors are actually used to contain the number, and the third integer contains the sign (0 for positive, -1 for negative).

For example the number 3467389200559205849930147290165382927106738 in vecthor form is:

{32, 2, 0, 32} *30 vectors w/ only 0s* {0, 0, 346, 73892005} {59205849, 93014729, 01653829, 27106738}

We also decided to code in C for raw speed and usability in C++ code.

2.1 Single vs. Multi-core Operations

When we started our project we wanted to make use of the Cell's communication abilities by using Direct Memory Access (DMA) to allow all six SPU's to work on a single operation at the same time. However this approach would have far more difficult to implement than expected and the constant data transfer required would have hurt the speed. So with expert advice from DeLesley Hutchins we decided to focus on creating operations for single SPUs, and planned to have the PPU's dole out jobs to each SPU.

Once we decided to make an operation run on a single SPU, we considered using the GNU Multi-Precision integer library (GMP) and modifying it to utilize the vector functions of the SPU. Unfortunately, we could not get GMP to compile for the SPU, presumably to the SPU's limited built-in functionality. We therefore proceeded to create a SPU-optimized library from scratch.

2.2 Data Type Development

In our original implementation, we dynamically sized the vectors so that they always had just enough vectors to contain the number (plus the meta vector). However, this meant that the vector had to be resized whenever the number overflowed the current vector, and a lot of time was wasted on memory allocation. For example, if advanced carrying was necessary in addition with the old system, for a function $c = \text{func}(a, b)$, first c had to be put in a temporary vector, then c had to be re-initialized to be one vector larger. Then the process had to be repeated as soon as the new vector was filled, which in the large calculations this is designed for was quite often. One call of the C functions `calloc` and `malloc` takes about 340 times as much time as an intrinsic add or subtract operation. With dynamic resizing, over 50% of the time for a single arbitrary-precision subtraction operation was used to allocate memory, and almost 50% of addition was allocating memory.

With a mostly fixed-size vector, overflow happens far less often – and the vector is simply doubled in size, which means that it's unlikely to overflow again any time soon and much less time and resources are wasted on resizing the vectors. A 2048 bit prime number key for RSA encryption is roughly 600 decimal digits. We chose a 32 vector vector as it holds 1024 digits, so for these smaller prime numbers this means we won't have to resize the vectors. Vectors now double in size when the numbers exceed 1024 digits, 2048 digits, etc and halve with getting smaller.

2.3 Limited Success

Because of the nature of the Cell, our initial goal was to have the PPU manage jobs for the SPUs, while the SPUs performed their jobs extremely quickly and reported back to the PPU.

Unfortunately, we quickly realized that programming at this low-level on a foreign processor architecture was unrealistic based on time constraints, and essentially beyond the scope of our project. As a compromise, we decided to run programs on the SPU alone. There are a few advantages to this:

- Writing code in C using vectors was much more easier to understand than working around the complexity of the DMA (Direct Memory Access) Transfers, individual mailbox routing to and from SPUs, and context/thread creation and management for each SPU.

- Using vectors on an SPU would give us a reasonable understanding on whether or not the Cell's philosophy and technology is really a breakthrough. Therefore, we could write and run code relatively quickly while still being able to draw a conclusion on the Cell as a whole.

- The SPUs are known to be the fastest part of the Cell. A comparison of the new, exotic SPUs to a normal intel processor would be much more beneficial than one of the power PC brand PPU to an intel processor.

No arbitrary precision integer library currently exists for the Cell, and if the processor's power was truly revolutionary, we may have seen dramatic increases in calculations with prime numbers, for example. We gave our library the name 'thor', and decided to answer the question everyone is asking: Are the speed benefits from the Cell *really* worth the hassle of programming on a completely unexplored, foreign processor architecture? Cleve Moler assured us that they are not, and as we discuss below, we have come to the same conclusion (at least for this application).

2.4 Function List

Below is a list of all the functions we created, with a description of how they work and how they are used. The full code is provided in appendix A.

```
typedef vector signed int* __attribute__((aligned(4096))) vecthor;
```

Summary: Defines the size of a vecthor.

```
vecthor init_vecthor(void);
```

Summary: Initializes a 4096-bit vecthor and returns a pointer to that vecthor

Description: Our function for initializing vecthors simply allocates 32 vectors, each vector consisting of 128 bits (4096 bits for a standard vecthor). This is done using the `calloc` function, which allocates and initializes a block of memory. The metadata is also appropriately initialized, and finally a pointer to this newly created vecthor is returned. Initially, we passed the desired size of the vecthor as the first argument. However, after we switched to a fixed-sized datatype implementation, this was no longer needed.

```
void vecthor_clear(vecthor a);
```

Summary: Zeros out a vecthor, setting all values to zero and updating metadata accordingly.

Description: This function loops through each vector in the vecthor, and inserts the scalar value of zero into each element of these vectors. The metadata is also adjusted for the altered values.

```
void vecthor_copy(vecthor a, vecthor b);
```

Summary: Copies all data from *a* into *b*.

Description: To copy data from one vecthor to another, we simply loop through the first vecthor, placing all vectors in the source vecthor into the destination vecthor. This memory copying is admittedly slow. However, if one was to simply set the two vecthors equal to each other (e.g. $a=b$), this would only set the pointers equal to each other, therefore causing all further operations performed on *a* also to be performed on *b*.

```
void str_to_vecthor(vecthor a, char *str);
```

Summary: Sets *a* equal to numeric data found in *str*.

Description: `str_to_vecthor` works as follows: First, loop through the vectors in *a* from right to left. For each iteration in the loop, convert the next 8 characters in *str* into an *int*, then place this in the relevant element in the vector.

void vecthor_write_virtual_size(vecthor a);

Summary: Determines the size of the number in the 4096-bit vecthor space and adjusts the metadata vecthor accordingly. Used primarily in other library functions

Description: In order to determine the variable virtual size of the data inside a vecthor (rather than the actual size), we look at each vector in the vecthor from left to right. If, at any time during this loop, the given vector's contents are *not* equal to 0, then the loop is terminated and the size is updated in the metadata. This loop must go from left to right, otherwise a series of 32 zeros or more could possibly be interpreted as the end of the data in the vecthor (therefore corrupting the virtual size).

int vecthor_virtual_size(vecthor a);

Summary: Returns the virtual size of the vecthor, i.e. the number of vectors containing data. Used primarily to optimize loops in library functions.

Description: Literally, this function returns the second element in the metadata vector. This variable can be used to optimize nearly every core arithmetic operation in the library. The reason for this is that if a vecthor consists of a 32 vector-long chain, but only 1 or 2 of these vectors hold actual data, then it is useless to loop through this empty data (to perform addition or subtraction, for instance). Using the `vecthor_virtual_size()` function, we could easily tell where a vecthor's data started and ended, and thus could loop through only the data that mattered when optimizing our functions.

int vecthor_actual_size(vecthor a);

Summary: Returns the actual size of a vecthor, i.e. the number of 128-bit vectors allocated to the vecthor.

Description: As opposed to *vecThor_virtual_size()* (see above function), the actual size of a vector is the amount of vectors allocated to the particular vector. It follows that in order to determine the number of bits a vector occupies in memory, this conversion can be used:

*memory_usage(a)=128*vecThor_actual_size(a);*

as a vector takes up 128 bits.

int vecThor_sign(vecThor a);

Summary: Returns the sign of a vector (1 if positive, -1 if negative, 0 otherwise).

Description: This function returns the third element of the metadata vector.

void print_vector(vector signed int a);

Summary: Prints a single vector (not a vector).

Description: This function is used in debugging, to examine the individual elements of a vector. Normally, *print_vecThor()* or *print_vecThor_debug()* is desired.

void print_vecThor(vecThor a);

Summary: Prints an entire vector to screen in an easy-to-read fashion.

Description: *print_vecThor()*, after determining the relevant size bounds of the vector, takes all data vectors and displays them on the screen. Extra precaution is taken in ensuring that the padding is correct (e.g. eight zeros will not be shortened to one, and zeros on the left will not be displayed).

void print_vecThor_debug(vecThor a);

Summary: Prints an entire vector to screen, including meta-data and zeros

Description: This function quite literally prints all data (relevant or not) that the vector occupies in memory onto the screen. This is used in testing functions, and is generally excessively difficult to read.

void vecThor_lightning_add(vecThor c, vecThor a, vecThor b);

Summary: As fast as the lightning of thor! Computes the sum of a and b , storing the result in c . This function does not carry for you, so it should only be used when it is certain that no overflow will occur.

Description: This function uses the assembly language intrinsic `spu_add()` to perform simple addition on two vectors. It is best used when incrementing a larger vector by a smaller amount, or for adding small values.

```
void vecthor_add(vecthor c, vecthor a, vecthor b);
```

Summary: Computes the sum of a and b , storing the result in c .

Description: Addition is performed using the same `spu_add()` assembly language routine, however, carrying (the heart of addition) is performed afterwards to ensure that the data is lined up correctly.

```
void vecthor_carry(vecthor c);
```

Description: Ensures that each element is 99999999 or below. Properly formats the vector. Use after any operation that increases the size of entries.

Summary: This function is a carry function for use in addition. It utilizes the fact that we only use 8 digits per vector entry while an int can go up to 9. It loops through each entry of each vector, dividing the entry by 100,000,000 (our biggest allowed int plus 1) to obtain the carry and calculating the entry mod 100,000,000 to obtain the “mod”. The carry is added to the next entry to the left, and the mod replaces what was previously in the entry.

```
void vecthor_carry_ext(vecthor c, int biggest);
```

Summary: Carries, but with the option of manually setting the biggest-int size

Description: Instead of treating the biggest int as 99999999, the biggest int is specified by the user.

```
void vecthor_sub(vecthor c, vecthor a, vecthor b);
```

Summary: Computes the difference of b and a , storing the result in c .

Description: Subtraction is first performed normally (e.g. simply subtracting everything in b from a , not worrying about negative values or borrowing). Next, we look at all negative values.

For each negative value, we:

- Take the absolute value of this number, making it positive.
- Subtract this positive value from our biggest int + 1 (100,000,000).
- Subtract one from the value to the left of it in the vector.

void vector_mul(vector c, vector a, vector b);

Summary: Computes the product of a and b , storing the result in c using the slow but straightforward naive multiplication. The product must be less than 1024 digits.

Description: Naive multiplication is very similar to multiplication by hand. We first loop through all the vectors in b and multiply each of these vectors by every vector in a . The result is stored in an *unsigned long long*. The result is split into two parts (the first part is the division by the biggest int, the second part is the long long mod our biggest int), and added to the appropriate 'slot' in the vector. Periodically, the result vector must be carried to prevent overflow.

void vector_pow(vector c, vector a, vector bb);

Summary: Uses successive squaring to compute a raised to the b power, storing the result in c . b must be less than 99999999.

Description: Successive (sometimes called repeated) squaring involves looking at the binary representation of b (the exponent). If the LSB is one, then our result vector is multiplied by our base. Regardless of b 's binary representation, our base is square with every iteration. At the end of each iteration, our exponent is shifted right. The loop terminates when the exponent is equal to zero.

void vector_mod(vector c, vector a, vector b);

Summary: Computes $a \bmod b$, storing the result in c .

Description: Uses an algorithm that shifts b to the left until its binary representation is one bit less in length than a . Then, b is subtracted from a until $a < b$. This is repeated until a is less than the original b .

void vecthor_powm(vecthor c, vecthor a, vecthor bb, vecthor m);

Summary: Computes a raised to $b \pmod{m}$ and stores the result in c using successive squaring.

Description: See *vecthor_pow()*. The algorithm is identical, except a is squared and subsequently modded by m with each iteration.

void vecthor_is_prime(vecthor c, vecthor a);

Summary: Uses Fermat's Little Theorem to test the primality of a . a must be less than 99999999. Splats the result vecthor with 1's if a is prime, 0's otherwise.

Description: Fermat's Little Theorem uses a small prime number as a base (2, 3, 5, 7), and raises this to $a-1$, modding the result by a . If the result is not 1, then a is definitely not prime. If the result is one, then we repeat the process with a different base. The more bases used, the surer one can be that a is prime.

int vecthor_comp(vecthor a, vecthor b);

Summary: Compares the size of a and b . Returns 1 if $a > b$, 0 if $a == b$, and -1 if $a < b$.

Description: This function loops through a and b , using assembly language intrinsics to determine if one vector is greater than another. If one is found to be greater than another, then the loop terminates.

int vecthor_cnt_lz(vecthor a);

Summary: Counts leading binary zeros in a vecthor.

Description: This function counts the binary leading zeros in a vecthor by using a loop coupled with the *spu_cntlz* function.

void vecthor_lshift(vecthor c, vecthor a, int times);

Summary: Multiplies vecthor a by 2 $times$ number of times using a left-shift intrinsic function and carrying, storing the result in c .

Description: This function calls *spu_slqw()* and carries when appropriate to prevent overflow or data discrepancies.

int vecthor_bin_diff(vecthor a, vecthor b);

Summary: Used in modding to determine how many times *b* would have to double to have the same size binary representation as *a*.

Description: This function simply subtracts the values of *vecthor_cnt_lz()*.

int vecthor_pos_extract(vecthor a, int pos);

Summary: Extracts the number in position *pos* from vecthor *a* where position is measured from right to left, (ie. the rightmost integer has pos=0 and the leftmost would have pos=32, since a standard size vecthor holds 32 integers). Used exclusively as an auxiliary function in multiplication.

void vecthor_pos_mul_add(vecthor a, int pos, unsigned long long x);

Summary: Takes *x*, which represents two ints multiplied, split it into parts greater than and less than 99999999 and add it to the values to existing values in *a*. Used exclusively as an auxiliary function in multiplication.

int vecthor_find_max_pos(vecthor a);

Summary: Returns the "size" of *a*, counting from left to right. Max is 32*4=1024. Used exclusively as an auxiliary function in multiplication.

void start_timer(void);

Summary: Starts SPU timer, measured in ticks. One tick is approximately equal to 40 clock cycles.

int read_timer(void);

Summary: Reads from SPU timer, returning the number of "ticks" since the timer started.

double ticks_to_ms(int x);

Summary: Converts x ticks to milliseconds, returning the number of ms for a given number of ticks.

void init_aux(void);

Summary: Should be called at the beginning of *main*. Initializes a number of auxiliary vectors used to extend the functionality of certain operations.

Description: This function simply calls *init_vecthor()* on a number of global auxiliary vectors. These are used when memory overwriting may occur (e.g. calling *vecthor_add(a,a,b)* may cause memory discrepancies).

int add_elements(vector signed int a);

Summary: Adds all of the elements in a given vector, returning the result.

int intpow(int x, int pow);

Summary: Computes x raised to the pow power. Used in *str_to_vecthor*.

int abs(int x);

Summary: Returns the absolute value of an integer using bit logic to avoid branching.

2.5 Software Used

To enable code writing and compilation on the Playstation 3 systems, we installed Fedora or Yellow Dog Linux distributions on each system. Recent firmware updates have disabled the option to install a third-party OS, though Sony has disabled and re-enabled this capability in the past. To compile C code on the Cell's PPU and SPU cores, we used Cell-specific gcc builds created by IBM named spu-gcc and ppu-gcc. These are available for free on the IBM website as part of the "Software Development Kit for Multicore Acceleration Version 3.0,"

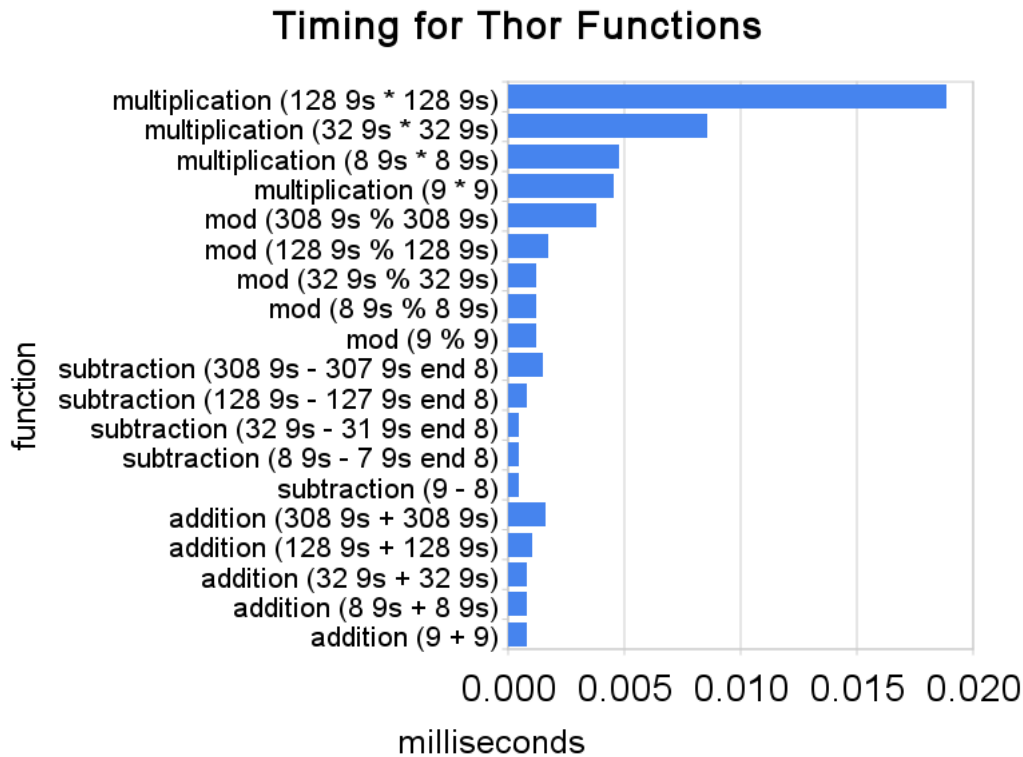
2.6 Literature Used

For instructional material, we used both books and IBM manuals for development on the Cell. For general information on programming in C, we used Aitken's *Teach Yourself C in 21 Days* and Kernighan and Ritchie's *The C Programming Language*. For Cell specific information, we used *Programming the Cell Processor*, by Matthew Scarpino. This was our

single best resource, without which we could not have produced as much Cell-specific code as we did. The IBM manuals we used were all part of the Cell SDK. Citations for these sources are provided at the end of this document.

3 Results

Below is a chart of the time taken for a single run-through of our primary functions. We tested each function with different numbers, once with 1 digit (e.g. 9+9), once with a full entry in a vector (99 999 999 + 99 999 999), once with a full vector (32 9s + 32 9s, etc), once with 4 full vectors, and one test with 308 digits which is roughly the length of a 1024-bit number, and therefore a component of a 2048-bit RSA key.

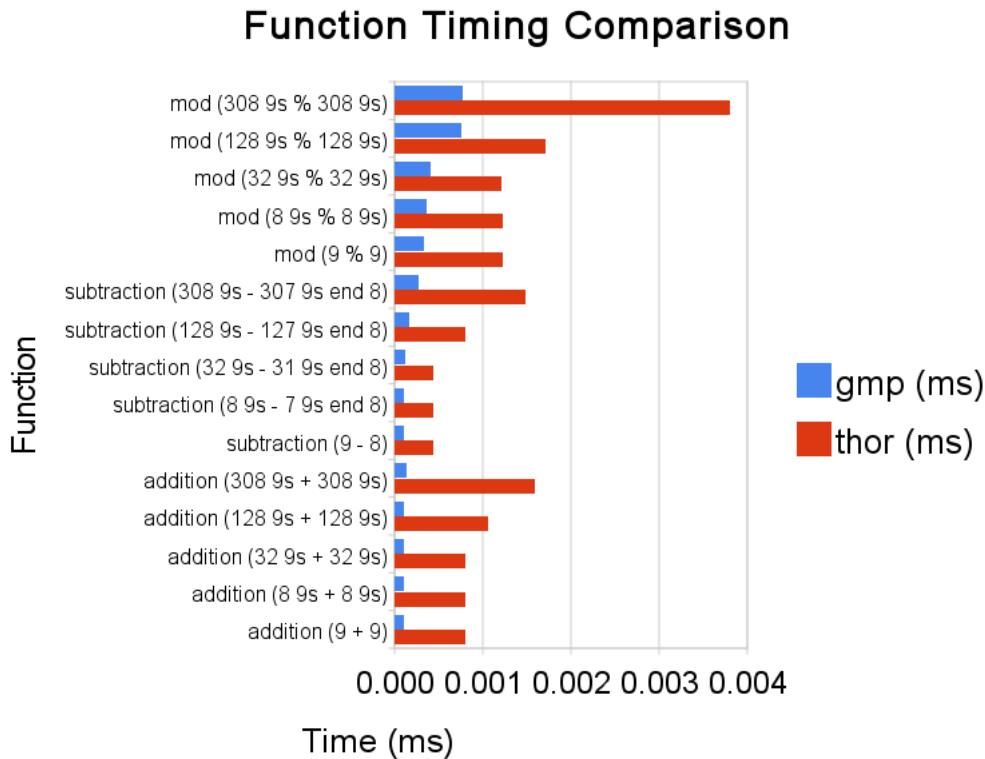


We measured the timing in SPU ticks, which is equal to 40 processor cycles. To get more comprehensible view of the time taken, we translated the values in ticks into milliseconds by dividing the values by 80,000 (3.2 billion cycles per second, 3.2 million cycles per millisecond / 40 cycles per tick = 80,000 ticks per millisecond).

3.1 Comparison to GMP

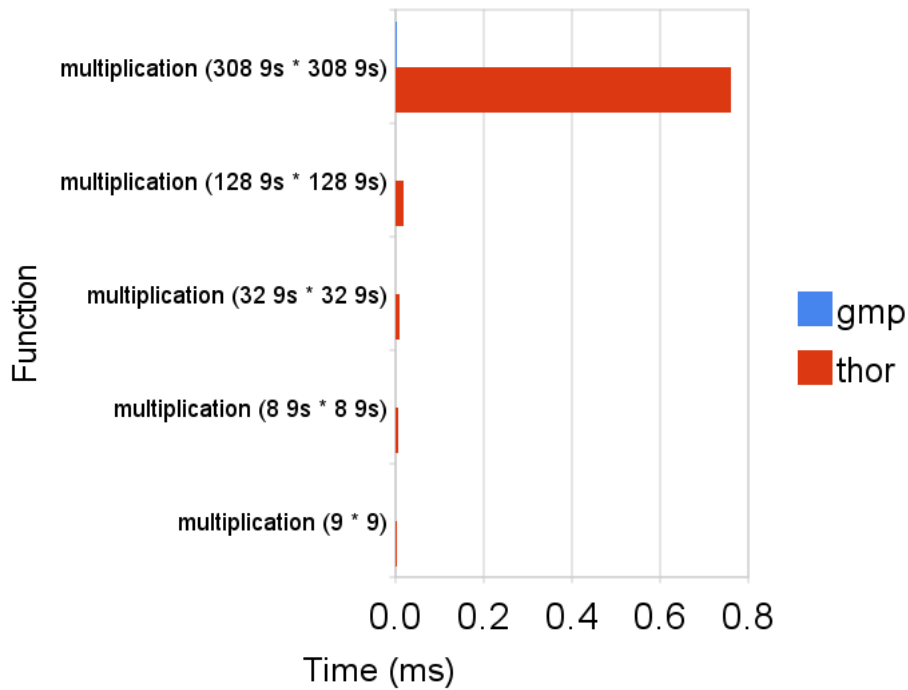
The real test came when we compared the timing of our functions to the industry standard of arbitrary precision arithmetic, the GMP (GNU multiple precision) integer library. For the GMP data below, we used the PPU, which runs at the same 3.2GHz clockspeed. Converting PPU and SPU results to milliseconds allowed us to compare them directly. Of course, different processor architecture is being used in each case, but as GMP will not compile on the SPU it is as close as we could get to a true comparison.

These graphs show that GMP's functions far outperform thor's functions.



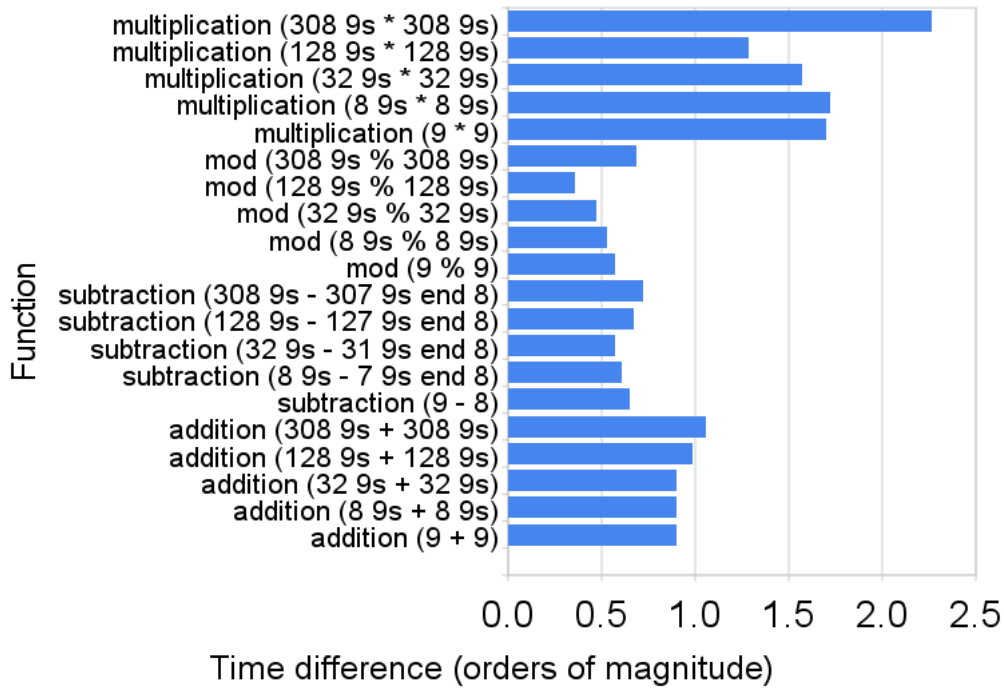
The timing below is correct. While our adding and subtracting functions likely use similar algorithms to those used in GMP, our multiplication is naive multiplication (the type you do by hand), and is therefore very slow compared to GMP's more sophisticated multiplication algorithms.

Multiplication Timing Comparison



While GMP obviously outperforms thor for reasons discussed below, most functions are within an order of magnitude difference. Multiplication is noticeably different, but as mentioned above, GMP utilizes Fast Furier Transform for multiplication, so we expected ours to be much slower. The comparison is like apples to oranges until we develop FFT multiplication for the Cell.

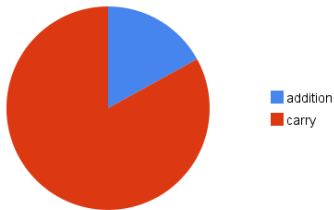
Time difference, in orders of magnitude



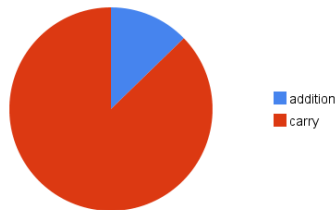
This graph shows the log of the ratio between thor's functions and GMP's functions. It displays the time difference between comparable functions in terms of order of magnitude (i.e. 10^2) to allow for better visibility of the data.

The following charts show the percentage of total time in an addition operation used in adding and carrying. The percentage of time used for carrying is roughly 83%, 87% and 91% respectively. For addition operations smaller than 9 digits, carrying takes up roughly 83% of the total time as well.

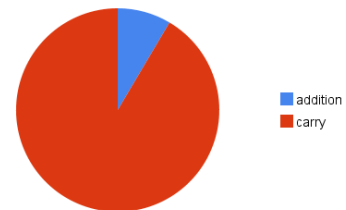
Percentage Carrying in Addition (9 + 9 through to 32 9s + 32 9s)



Percentage Carrying in Addition (128 9s + 128 9s)



Percentage Carrying in Addition (308 9s + 308 9s)



4 Analysis

While we thought the Cell's novel structure would prove to be faster than a normal computer, the results proved just the opposite. The fact that SPUs are not good at branching

coupled with the fact that datatypes can not be directly manipulated by C lead our library to be much slower than we thought it would be. When compared to the GNU Multiple Precision library (GMP), we found that in several cases we were many orders of magnitude behind. While the Cell processor may sound good on paper, vectors may be more trouble than they're worth, at least for integer arithmetic.

4.1 Parallelizability / Comparison to GMP

The Cell processor is praised for its ability to be parallelized and optimized for working with four integers at once (vectors). However, when it comes to integer math, algorithms are only partially parallelizable. For instance, when carrying after performing an addition operation, every significant bit relies explicitly on the bits less significant than it, and the algorithm must operate from right to left. Thus, even with 6 or 7 SPUs, addition can not be optimized for multiple processors. This does not lie inherently in the Cell processor, but is rather a weakness of positional notation in general. Modulus and multiplication do not parallelize well either: our implementation of modulus is reliant on the results of repeated subtraction and shifting, and multiplication still involves the implementation of a carry method. Because of this, arbitrary precision libraries tend to be inherently slow, and some algorithms *must* be linear in fashion. Even if the core of the operation can be parallelized, the carry must still be done linearly. This greatly stunts the ability to optimize algorithms such as these.

Another reason GMP outperformed thor was because of the difficulty of memory access on the Cell. On a modern CPU, one can simply execute `a[2]+=10;`. The processor is already adept at dealing with arrays of numbers. However, on the Cell, that piece of code would resemble this: `*a=spu_insert(spu_extract(*a,2)+10,*a,2);`. This is actually three operations, each of which takes about 3 CPU cycles. The Cell must copy an integer from a place in memory, add to it, and insert it again into memory. This lengthy bit of code leads to more compiled instructions, less efficient memory management, and ultimately less efficient code. This leads us to conclude that when it comes to raw computing power in relation to small individual operations, the Cell is definitely not the weapon of choice.

We expected our addition and subtractions to be most comparable to those utilized by GMP, because they are the most obviously and simply parallelizable. Indeed, the actual adding or subtracting is remarkably easy. However, carrying involves several operations for each

element of each component vector:

Step 1: extract an element from a vector, mod it by `BIGGEST_INT + 1`

Step 2: add the result to the next entry over

Step 3: divide the extracted element by `BIGGEST_INT + 1`

Step 4: Insert the result of that operation back into its original place

In C, it looks like this, where `*(c+i)` is the entry being processed

```
temp=spu_extract(*(c+i),j);
*(c+i)=spu_insert(spu_extract(*(c+i),j-1)+temp/(biggest+1),*(c+i),j-1);
*(c+i)=spu_insert(temp%(biggest+1),*(c+i),j);
```

If the Cell had built-in intrinsic functions designed for integer modding and integer division, this process would be much faster. Unfortunately, it does not. As it is, this sequence of operations must be done for every single entry of a vector. We suspect that this carrying operation is what makes our adding algorithm slower than GMP's, not the adding *per se*. As it was, results indicated that 80-90% of the CPU cycles in an adding operation were used for carrying. If carrying were able to see the same 4x increase that pure addition does, an operation that now takes 100 time units would take roughly 28 time units. This would make our operations take 3x as long as they do using GMP, rather than 10x.

With that said, a well-parallelizable, higher level algorithm may in fact see a speed benefit from the Cell architecture. If work was divided up in an efficient way and doled out to the SPUs intelligently, each one could perform their small bit of the problem sequentially (the part of the problem would have to be large enough as to not suffer from the above speed penalties). For example, when generating a large prime for use in an RSA key, each SPU could be asked to look at a different set of numbers, meaning the overall result would be found 6x as fast as using one SPU. However, in this particular example, since GMP cannot be compiled on the SPUs and that is not fast enough to be worth using, a standard multi-core processor using GMP is still a better choice. Given the successful application of the Cell in physical simulations and multimedia, it seems the Cell is great for floating point computations that are not strictly

sequential. However, the Cell does not show appreciable value under the constraints of integer arithmetic. The above considerations also suggest that using highly parallelized architectures such as GPUs or upcoming GP-GPUs for doing integer math would probably not be worthwhile.

5 Conclusion

Through exploring the capabilities and limitations of the new Cell processor architecture, we have learned not only of some specific drawbacks to using the Cell, but some drawbacks of parallelized computing in general. For example, while vectors may be able to hold four integers and operate on them simultaneously, the strenuous task of reading from and writing to these vectors when operating on them is slow. Also, when performing numeric operations, because of the nature of arithmetic algorithms, some pieces of data rely on other pieces. This leads to a large stunting in the amount of data that can be parallelized at a low level, and, therefore, a smaller amount of potential in multicore programming to be realized.

The Cell's SPU cores reputedly lack efficient branching capabilities, which also greatly hurt us. Many arbitrary precision libraries examine the input and select an algorithm based on length of the operands. On the Cell however, determining which algorithm to use on specific input data would be extremely time consuming, and this limited our speed greatly. Keep in mind that we ran into branching problems on algorithms as low of a level as adding. A higher level algorithm might experience even more severe impediments, depending on its design.

On a larger scale, we can safely conclude that when examining a program with the intent of parallelizing it, one should not necessarily attempt to distribute the low-level number crunching loops, but rather examine the algorithm from a top down perspective and optimize it in this fashion. However, the textbook way of optimizing a program for multiple cores typically says the opposite. One is usually told to look for bottle-neck loops in the program and divide these up between available processors. From our extensive experience on the Cell, we can say that this is in fact *not* the way to go about it. The main reasons for this are the Cell processor's sheer incompetence when it comes to completing low level sequential integer operations smoothly. Because these low level operations require a large number of functions to accomplish relatively simple tasks, the Cell benefits from a top-down optimization approach. Most of our optimization mistakes were made in assuming that the best way to optimize was to tweak functions at as low of a level as possible.

5.1 Most Significant Original Achievement

Our most significant achievement is taking several algorithms we take for granted when using high level languages (addition, subtracting, multiplication, string and memory manipulation), and examining them closely at the lowest possible level. It has taught us a great deal about what it was like to implement the algorithms, as well as specific details as to how compilers, bits in memory, memory allocation, assembly language, threading, processor architecture, and intense line-by-line optimization works. The final product, `spu_thor.h`, may or may not be used extensively by the Cell programming community (or lack thereof). However, through producing the library we have learned more than ever before about the nature of computers and algorithms, and it will continue to influence the way we meticulously build and optimize our programs.

6 Resources

Aitken, Peter G., Bradley Jones, and Peter G. Aitken. *Sams Teach Yourself C in 21 Days*. Indianapolis, Ind.: SAMS, 2000. Print.

"DeveloperWorks : Cell Broadband Engine Resource Center." *IBM - United States*. Web. 07 Apr. 2010. <<http://www.ibm.com/developerworks/power/Cell/>>.

Kernighan, Brian W., and Dennis M. Ritchie. *The C Programming Language*. New Delhi: Prentice-Hall of India, 1999. Print.

Scarpino, Matthew. *Programming the Cell Processor: for Games, Graphics, and Computation*. Upper Saddle River, NJ: Prentice Hall, 2009. Print.

We would also like to give special thanks to the Los Alamos National Lab Foundation for providing the funding used to purchase the equipment used in our project. Without their help, this project would have been difficult to impossible. Thank you!

The Spread of the Black Death in London

New Mexico

Supercomputing Challenge

Final Report

April 7, 2010

Team 37

Desert Academy

Team Members:

Katie Boot

Sara Hartse

Teachers:

Thomas Christie

Jocelyn Comstock

Table of Contents

1. Executive Summary	3
2. Introduction	4
A. Goal	
B. Hypothesis	
3. Description	5
A. Biology	
B. Background	
C. Significance	
D. Model	
4. Development	9
5. Results	11
6. Conclusion	15
7. References	18
8. Code	19

1. Executive Summary

The objective of our project is to model the development of the bubonic plague in London for the duration of the primary outbreak in 1347 that lasted into 1350. We are applying epidemiology to the pandemic and analyzing the consequence it had on general social classes, for example the peasant's mortality rate versus that of the nobles and higher classes.

We chose this topic because we are interested in epidemiology, and wish to know how and why past societies were affected by circumstances that can be encountered today. We are also curious about the spread of the bubonic plague in past communities versus its spread in modern communities. If we can accurately replicate the Black Death, we could continue with the project and model a version of the Black Death in the modern-day London community. This would enable us to compare the spread of the plague in modern times to that of the past.

While our final model does not reflect the degree of complexity we originally intended, it takes into account the chances of infection, recovery and death, the geometry of 14th century London and the fact that the disease was initially brought to London by ships. We were able to determine model parameters, such as probability of recovery and the infection rate, that provide model results which correspond to historical data to a relatively high degree. This information will allow us to continue to use this model and add more elements in the future.

2. Introduction

A. Goal

Our goal for this project is to create a model of the bubonic plague in London and have it produce historically accurate figures, specifically with regard to the rate of death and overall rate of infection. We should be able observe the effect of the plague and its variations in different sectors of London. We ultimately want this model to be used to compare the effect of the plague on different social classes, comparing the mortality rates of peasants to aristocracy, or different professions such as the clergy or farmers.

B. Hypothesis

We predict that if we successfully create and calibrate our model so that it gives historically accurate results, then we will be able to see trends of people of higher class surviving longest and members of the clergy dying first. Furthermore, we predict that agents who start off in more crowded areas, especially near the Thames, will die more swiftly than those in more isolated areas.

3. Description

A. Biology

The bubonic plague, also known as the Black Death in the mid-fourteenth century, is derived from a bacterium called *Yersinia pestis* (*Y. pestis*), and is spread by the bite of an infected flea, *Xenopsylla cheopis* (the rat flea). The term “bubonic plague” comes from the most recognizable symptom, buboes, which are lymph nodes that become inflamed and swell to a substantial size. Victims of the plague generally die within three to seven days of infection. The reason for the plague’s lethality is that humans rarely have immunity to it, and once contracted, the plague usually spreads too quickly throughout the body for one’s immune system to react to it. However, if a person survives the plague they would probably never contract it again.

The bubonic plague cannot disseminate directly from person to person, but it can develop into two other forms of plague, septicemic and pneumonic. Septicemic plague is developed through blood poisoning, and cannot spread from person to person. Pneumonic plague, derived from pneumonia, is communicable through people and can circulate quickly through a populace by means of coughing, and can cause death in three days or less. During the Black Death, the pneumonic plague worked with the bubonic plague and caused approximately half of the total plague-related deaths.

Though the bubonic plague can be detrimental to a human population, it usually only occurs in rodents. This disease is only dangerous when it mutates and breaks out of its out biological group (rodents) and infects other groups (humans). Fleas spread the illness from infected rats to healthy rats by biting them and transferring some of their blood. Rats can transfer the plague to humans by biting them, in which case the plague is administered directly into the body.

Rats are the preferred host of the *Xenopsylla cheopis*, but if the rat population happened to decline, the fleas would be forced to find new hosts such as humans and livestock. The plague can also spread when the bubonic plague mutates into the pneumonic plague, which can move from person to person.

B. Background

Thought to have originated in East Asia, the outbreak of the bubonic plague was devastating for the European population. It was the first major disease to reach Europe in centuries. The plague first appeared in Europe in 1347 and swept through the populace for the next two and a half years, killing over 25 million people CITE – where is this data from? With a preliminary population of approximately 7 million, England's inhabitants declined as the plague killed almost half of its citizens. All of the social classes were affected, though the peasants were the most susceptible due to unhealthy living conditions and overpopulation. Only a few members of the nobles and royal family died due to the plague. The loss of the peasant class caused a great decline in food production, which contributed to the famine already sweeping the countryside, thus killing more people.

The people of the middle ages had no effective way of treating the plague and it was not until antibiotics were developed that there was any way of stopping it at all. People blamed the plagues outbreak on a couple factors, including "bad air," witches, astrology and a rare alignment of planets. Many people believed the Black Death signaled the end of the world, or the apocalypse. Others thought that the Jews had created the plague as a way to destroy the Christian world. Thousands of Jews and other minorities were killed and tortured by the panicked masses of Europe, especially in England. England was already experiencing its own hardships when the

plague hit in 1347. The various harvests had been almost completely destroyed by rains, winter was approaching, and the lower classes, such as peasants, were slowly starving to death.

The plague hit the hardest in large cities, of which London is a prime example. The plague reached London through the rats that inhabited trading ships carrying goods from Asia. Once the ships docked, the rats and fleas dissipated into the city, infecting both humans and rats as they went. London was overwhelmed by a combination of pneumonic and bubonic plague. Nearly 50% of London's population succumbed to the plague, and thousands more died of starvation and other causes. By late 1350, the plague had subsided, but outbreaks would continue for the following three hundred years. It was not until the mid-1600s that the plague would be mostly eradicated.

C. Significance

The Black Death had many consequences and produced many changes that would make a huge difference in European life. It was a turning point of the development of human civilization. For example, it led to the decline of the religious dogma that had controlled most societies for centuries. It caused people to have a greater interest in the study of science and medicine, which continued to philosophy, art, and a new era of invention. Trade expanded and eventually more efficient trade routes were searched for, leading to the discovery of America. The epidemic also eliminated serfdom in much of Europe. After a great quantity of the lower class peasants was killed by the plague, peasants were no longer thought of as personal property, but as individuals, and necessary for a society to flourish. Although the Black Death had many negative results, it brought about an adjustment in lifestyle that changed the way people thought and behaved, and eventually, the course of civilization itself.

The plague has emerged in the human population quite a few times throughout history, but the most famous and widespread outbreak is the epidemic that occurred from 1347 to 1350. This is the outbreak on which our project is based. The bubonic plague is still a common problem in the world today, and is currently the cause of major epidemics in regions such as Uganda, Kurdistan, and northern India. Also, as of late, bubonic plague has been found increasingly in the United States, especially the Southwest.

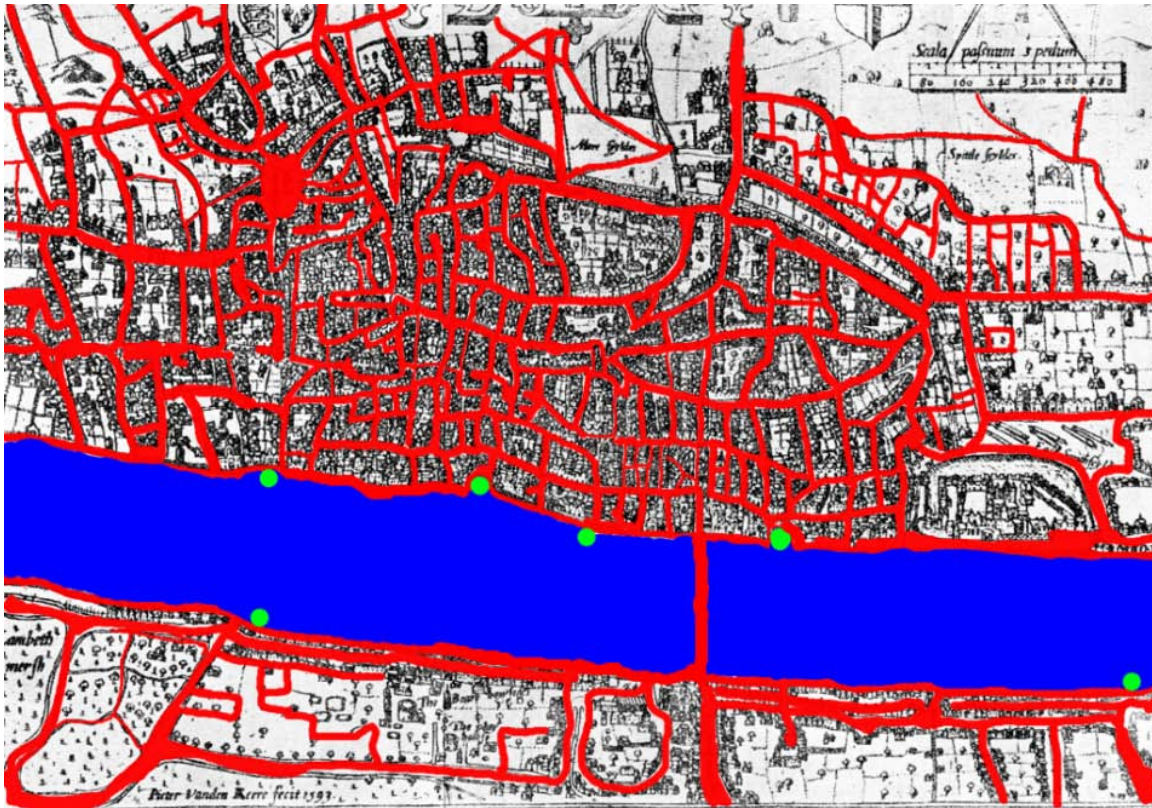
D. Model

We used Netlogo to model the spread of the plague in London. The basic function of our model is to create infected agents (rats), who start off at points along the Thames River. They dissipate randomly into “London” staying on specific paths and interacting with the human agents. There are about 7,000 human agents, each representing 10 of the 70,000 people living in London at the start of the Plague.

The sick rats are orange and they disperse through the London docks and infect the human and rat inhabitants of London through proximity. Healthy humans are green at first, and become yellow once they are infected. The infected agents generally live three to six more “ticks”. Human agents do not spread the plague as quickly as the rats do, and can only infect each other pneumonic plague, which has a lower chance of transmission. If a human or a rat manages to recover (this is moderately unlikely), then they become immune and the agent is shown in grey.

4. Development

Throughout the course of this project we learned a lot about epidemiology as well as NetLogo, the program we used to model the Black Death. First, we created a basic epidemiology model. After we had agents infecting and recovering, we added some factors that are more specific to the plague. For example, we changed the model so that rats were the original infected agent and the spread of the sickness depended on their movement and behavior.



One of our main challenges was incorporating a map of London into the model. We tried many different things, until we eventually colored in a fundamental map of fourteenth-century London using the GIMP photo-editing program. We colored in as many streets as we could, making sure that all the major roads and public areas in London were covered. We used red for this task, giving specific paths for the agents to move along and make contact with each other. We colored the Thames River blue and the docks green, and imported the patches from the

image into our model. This allowed us to make the agents responsive to the patch colors. The next step was to enable the “human” agents to move along the roads and remain on the red patches, avoiding the blue Thames River and the green docks. We used a piece of code that asserted when the setup button is hit the “human” agents automatically determine the color of their current patch. If the patch color is not red (the color of the roads) then the agents continue to move randomly until they move on to a red patch. We used a similar procedure to have the "rat" agents start off at specific colored points, namely the docks, which are the green patches in the model. These were placed at all the major docks along the London section of the Thames River, where the infected rats exit ships, and spread the plague among the human populace.



This is a screenshot of the model in the earlier stages of the epidemic.

5. Results

A large part of the modeling process has been tweaking various features of our model in an effort to make it as accurate, yet as functional as possible. Adjusting the size of the interface was a delicate matter; at one point there were so many pixels in “London”, that none of the agents ever had a chance to interact. We fixed this by shrinking the number of pixels and adding a variable that increased the infection radius, expanding the area surrounding an agent in which they could infect someone else.

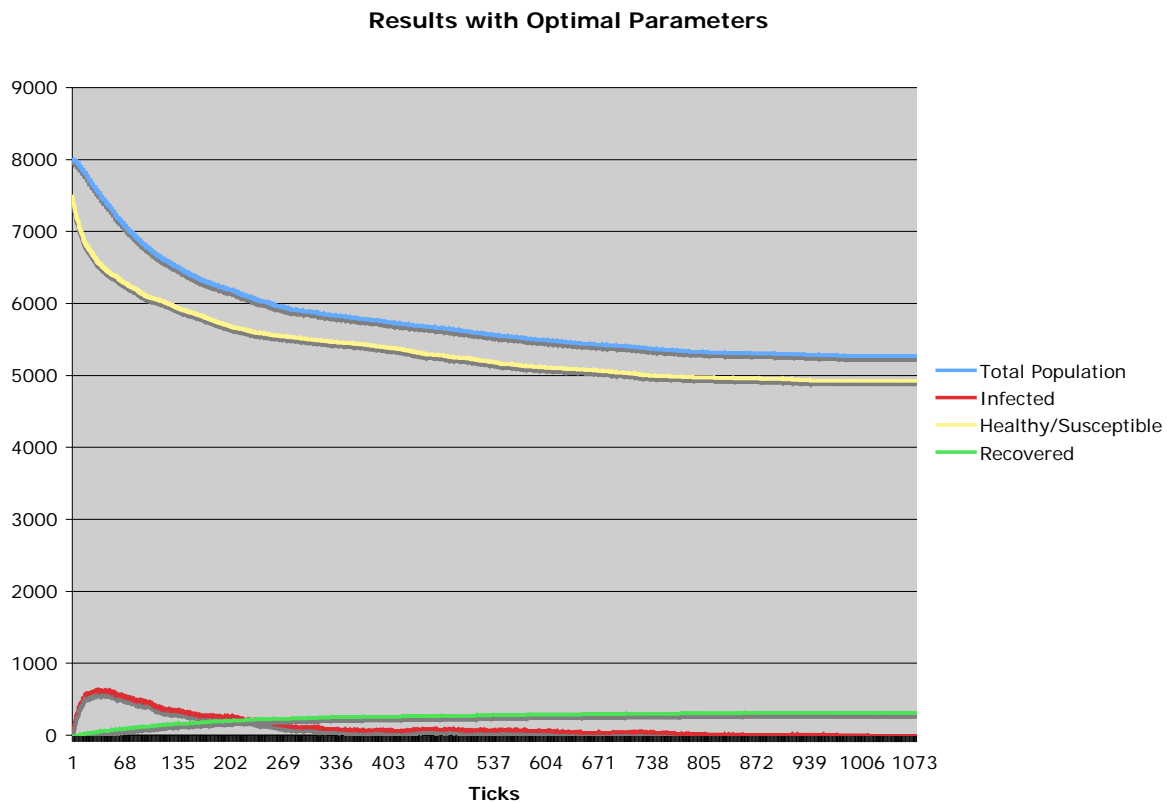
The other variables that have taken more time to adjust are those of the chance of dying, chance of recovering, the infectiousness of the bubonic plague and the infectiousness of the pneumonic plague. Our ideal model ends with about 5% of the population recovered and immune, 45% dead and the remaining 50% still susceptible. To figure out the perimeters that would lead to these results we have repeatedly run the models using the “Behavior Space” feature of NetLogo over long periods of time, altering these variables in increments of varying sizes each time. In total, we ran the model approximately 4500 times, each time with varying parameters. We then analyzed this information in spreadsheet form and found the ideal values for these variables.

The following chart shows the closest values for the variables that we found after repeatedly running the model. None of the calculated variables were ideal. We were aiming for a survival rate of 50 to 60% and the closest runs were still 70%. Our reasoning for this is that the increments we adjusted the variable in had to be fairly large in order to keep the number of times we had to run the model down to 720 times per computer (we used 6 different computers overnight). These numbers all produced survival rates that were too high, but we were able to take the values for the variables and adjust them to create a model that produced results closer to

the actual statistics of the Bubonic Plague in London.

Run number	Infectiousness-pneumonic	Chance recover	Chance die	Infectiousness	Steps	End Count		%
212	65	2	9	15	498	6214	Survived	77.7
212					498	5971	Never infected	74.6
212					498	243	Recovered	3.0
220	65	2	9	95	480	6113	Survived	76.4
220					480	5852	Never infected	73.2
220					480	261	Recovered	3.3
454	70	2	9	35	570	6117	Survived	76.5
454					570	5871	Never infected	73.4
454					570	246	Recovered	3.1
700	75	2	9	95	513	6016	Survived	75.2
700					513	5755	Never infected	71.9
700					513	261	Recovered	3.3
935	95	2	9	45	924	6073	Survived	75.9
935					924	5821	Never infected	72.8
935					924	252	Recovered	3.2
936	95	2	9	55	501	6054	Survived	75.7
936					501	5775	Never infected	72.2
936					501	279	Recovered	3.5
938	95	2	9	75	470	6019	Survived	75.2
938					470	5772	Never infected	72.2
938					470	247	Recovered	3.1
939	95	2	9	85	399	6013	Survived%	75.5
939					399	5765	Never infected	72.1
939					399	248	Recovered	3.1
940	95	2	9	95	729	5738	Survived	71.7
940					729	5463	Never infected	68.3
940					729	275	Recovered	3.4

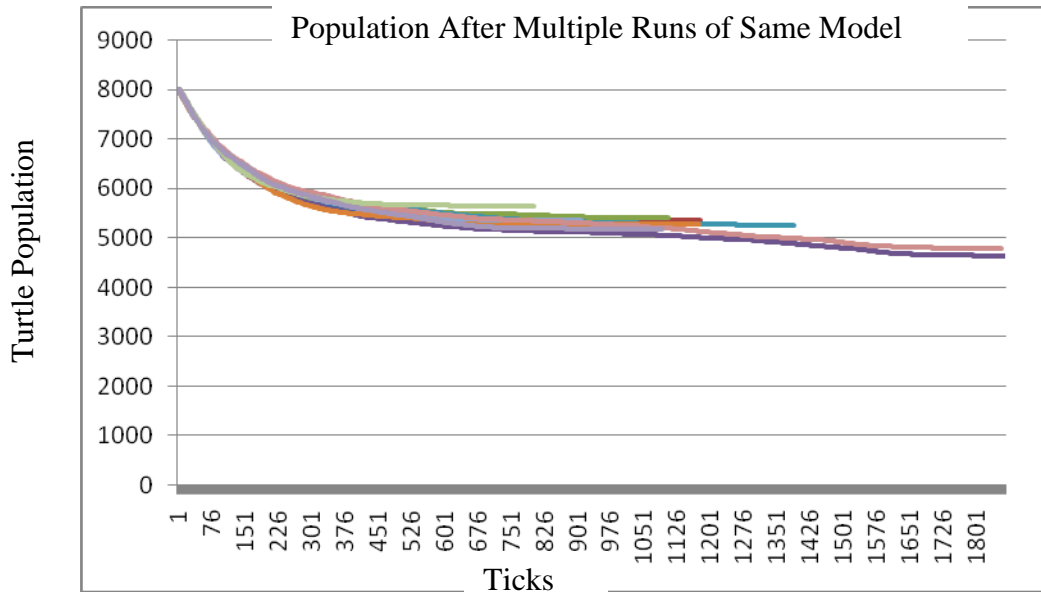
The variables that seemed to produce the most realistic results were infectiousness-pneumonic = 60, chance recover = 1, chance die = 5, infectiousness = 95. We decided to decrease the chance-die and chance-recover because the agents were dying so quickly that they could not move far enough to spread the sickness. Here is a NetLogo graph of the populations of the agents in our model while running it within these parameters.



Infectiousness-pneumonic = 60
 Chance recover = 1
 Chance die = 5
 Infectiousness = 95

After running the model with in these parameters many times, we began to notice patterns within the behavior of the agents and how they interact with their environment. Because of the paths that agents have to follow, they take longer to spread the illness. Some agents who are on wider paths come into contact with many other agents and if they happen to be near the

river, these are the first areas to be wiped out. The agents that live the longest are those that start out on smaller streets, farther away from the river. Because of this variation in terrain and the randomness of the agents' movement, different trials of the same parameters often produced slightly different results. The following graph records the populations of the agents over ten different runs.



6. Conclusion

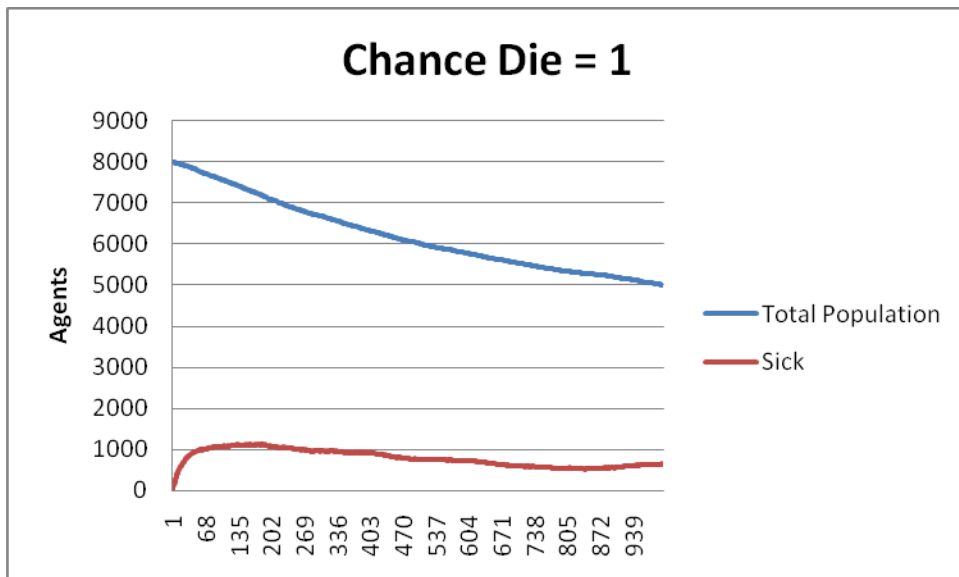
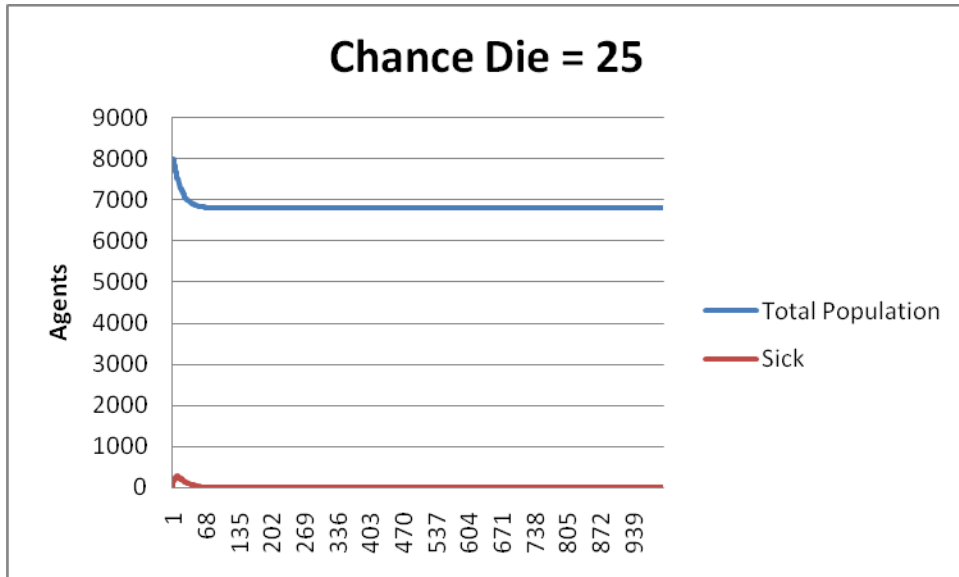
We were not able to get as far with our project this year as we had planned. As explained earlier, we managed to create a working base model of London and the plague, but we were not able to go into as much detail as we had originally intended. We did not finish making different agent behaviors to represent social classes. However, we were able to learn a lot about the epidemiology behind the Bubonic Plague and how the landscape of London affected that.

Our hypothesis that agents in wider, more crowded areas, especially those close to the Thames would die more quickly was correct. Agents in these areas transmitted the disease very quickly, causing the count of sick agents to spike. We observed that it only took a few infected agents to enter one of these areas and almost everyone would become sick. This was also true for very isolated areas, but these areas also had the highest rate of survival, because they had such limited interaction with the other inhabitants of the model.

While we were trying to correctly calibrate the model and pick values for the variables that would generate historically accurate results, we learned quite a bit about epidemiology in general. We learned that a higher chance of death was not necessarily the way to make an epidemic worse; high chance of death, at least in our model, caused almost all infected agents to die before they could infect anyone else. We discovered that the key to a long lasting and dangerous epidemic is relatively low chance of death and very high infectiousness. This creates a disease that, like the Bubonic Plague, is very contagious, but does not kill all of its victims immediately, allowing it spread it to many other people. This is shown in the following graphs.

In these two graphs, the first is one in which the chance of dying was very high. The second had much lower chance of dying, but despite the low chance of death about 2,000 more people died in this simulation than in the previous one. The high chance of death in the first run

meant that infected people died very quickly, before they could infect anyone else. The number of infected people spiked rapidly and then bottomed out. The lower chance of death in the second run kept the sickness around much longer, infecting many more people, eventually leading to more deaths. When calculating the parameters for our final model we actually had to raise the chance of death to have a higher survival rate.



Our final model is not entirely realistic, in that it is still missing a few key factors. The main way that we think it could be improved is by adding an incubation period for the agents. A period of time in which they are contagious but are not showing symptoms would greatly help us achieve our goal of realism. We could also improve the circulation of the agents to make the spread of the plague more realistic. We could do this either by making people walk on “tracks” instead of randomly “wiggling” through the streets, or by making them walk further between each proximity check.

There are many ways that we can expand and use this model in the future. The most immediate thing we would like to do is complete our goals for this year by adding different types of “human” agents to represent the different social classes with different movement patterns and trying to model the varying effects of the Bubonic Plague on them. We are also interested in modeling the bubonic plague on a much smaller scale, looking at the interaction between *Yersinia pestis*, *Xenopsylla cheopis* (the rat flea), rats and humans, as well as what causes *Yersinia pestis* to break out of its biological group of rats and move on to infect humans. What we accomplished this year has given us a good base model of the Bubonic Plague in London and a good point at which to start further investigation.

7. References

"1320: Section 6: The Black Death." *Welcome to Utah State University*. Web. 15 Jan. 2010.

<<http://www.usu.edu/markdamen/1320Hist&Civ/chapters/06PLAGUE.htm>>.

"The Black Death in England 1348-50." *UK Travel and Heritage - Britain Express UK Travel*

Guide. Web. 15 Jan. 2010. <<http://www.britainexpress.com/History/medieval/black-death.htm>>.

"The Bubonic Plague." *Essortment Articles: Free Online Articles on Health, Science, Education*

& More. Web. 15 Jan. 2010. <http://www.essortment.com/all/bubonicplague_rvdr.htm>.

"CBRNE - Plague: EMedicine Emergency Medicine." *EMedicine - Medical Reference*. Web. 15

Jan. 2010. <<http://emedicine.medscape.com/article/829233-overview>>.

"CDC Plague Home Page - CDC Division of Vector-Borne Infectious Diseases (DVBID)."

Centers for Disease Control and Prevention. Web. 15 Jan. 2010.

<<http://www.cdc.gov/ncidod/dvbid/plague/index.htm>>.

When creating this project we referenced several models in the NetLogo model library including Virus and African Plains, though the code in our model is all written by us. We would like to thank all the people who helped us and gave us feedback throughout the course of this project; we could not have done it without them.

Enlightenment: Metropolis-Hastings Ray Prediction Model in 3D Space

New Mexico
Supercomputing Challenge
Final Report
April 1st, 2010

Team #69
Los Alamos High School

Team members:

1. Kathy Lin
2. Jake Poston
3. Ryan Marcus
4. Doc Shlachter

Teachers:

1. Leroy Goodwin

Project mentors:

1. Leroy Goodwin

Table of Contents

Enlightenment: Metropolis-Hastings Ray Prediction Model in 3D Space.....	1
Executive Summary.....	3
The Rendering Equation.....	4
Monte-Carlo Methods.....	5
Description and explanation.....	5
Supercomputing.....	6
Ray Tracing Basics.....	7
Description.....	7
Problems with ray tracing.....	7
An overview of shading.....	7
3D Calculations.....	9
Defining Three-Dimensional Objects.....	9
Spheres.....	9
Rotation for cylinders and cones.....	11
Cylinders.....	12
Cones.....	14
The Metropolis-Hastings Light Transport Algorithm.....	17
Introduction and disclaimer.....	17
Explanation of the algorithm.....	17
Image comparison.....	19
Parallel advantages of the algorithm.....	20
Video produced by the algorithm.....	21
Conclusions.....	23
Code.....	24

All images depicting rendering were generated with Team 69's implementation of the algorithm, and used no third party rendering software.

Executive Summary

For years, the rendering equation has posed an unbreakable enigma to the scientific world. The sheer computational complexity of light – from diffraction to reflection to diffusion – forms one of the greatest problems known to physicists and computer scientists. For example, consider the light illuminating this sheet of paper: an uncountable number of photons are streaming down from a light source and striking this sheet of paper. After they hit this sheet of paper, some may reflect back into your eyes, and some may reflect out into space. Some may be absorbed into the paper, others may diffuse against the surface, and others still may pass through the sheet of paper and strike the desk beneath you. A complicated problem, to be sure – but one with a rather intuitive and simple solution.

Through a combination of upper-level statistical theory and brute force, the laws of probability can provide the solution to this complex problem. Our unique implementation of the Metropolis-Hastings algorithm gives us a method to predict light rays based on the path of other light rays. Through this method, we can compute a few rays and then use this method to discover all the other rays.

Our process begins by creating a scene containing a camera, a light source, and some objects. We then test various paths from the camera out towards objects to determine if the camera is looking at something that is illuminated. Once we find a few rays (paths from the light source to the camera), we create a sampling distribution based on properties of those rays. From that distribution, we randomly generate new rays that are then tested for accuracy. If any given new ray is found to be accurate, it is added to the sampling distribution.

This method allows us to create very realistic images at a very efficient rate. The creation of these high-quality images can be utilized for a number of purposes, ranging from analyzing X-rays to modeling light itself.

The Rendering Equation

Essentially, the rendering equation is the formal mathematical statement of how much light is emitted from a given point given the incident angle of the light, a given viewing angle, and various properties of the material (such as luster, reflection, index of refraction, etc). When one talks about “solving the rendering equation,” one does not speak of finding an algebraic solution to this formula. When considering solutions to the rendering equation, one looks at various methods that could produce the answer the rendering equation would yield. Actual mathematical manipulation of the equation itself would prove fruitless because many variables are never known, even in completely simulated situations.

Even though the equation is incredibly general, it still does not properly account for several aspects of light.

1. Fluorescence: When light bounces off an object (reflection or refraction) and has a different wave length then when it first hit the object.
2. Interference: When light waves constructively or destructively interfere with each other, such as in a double-slit experiment (described here: http://en.wikipedia.org/wiki/Double-slit_experiment)
3. Phosphorescence: When light is absorbed and not immediately emitted, such as glow-in-the-dark shirts or shoes.
4. Surface Scattering: Because the rendering equation (and almost every subsequent rendering algorithm) assumes that, with enough depth, every surface is entirely smooth, some objects may look unnaturally solid.

Because these constraints are built into the rendering equation and any given computer rendering algorithm is an attempt to provide the solution to this equation, no strictly-traditional rendering algorithm will take these into account either. The method documented in this paper, however, will (optionally) compensate for fluorescence and surface scattering.

Monte-Carlo Methods

Description and explanation

Often, a perfect mathematical solution is not available for a given problem. For example, the integration of many functions (like the normal distribution) can only be estimated. One tool in the mathematician's arsenal for resolving these troublesome situations is the Monte-Carlo method. Implementations of Monte-Carlo methods involve taking a large number of random samples from some form of distribution relating to a given problem, and then applying those samples in such a way as to estimate the actual solution.

Imagine that a mathematician is given a pair of two-dimensional closed shapes that exist across a single known domain and range. The only thing that the mathematician can test is whether a given point lies within a given shape. The mathematician wishes to determine which one has a greater area. For now, consider that the domain of both shapes is $[x, y]$ and the range is $[a, b]$. If this mathematician wished to employ a Monte-Carlo method, s/he would take a simple random sample of points within $[x, y]$ and $[a, b]$ for both shapes. After determining a finite number of points, the mathematician could conclude that whichever shape had the lower number of points within it had the smaller area.

Two uniform properties of Monte-Carlo methods make them especially applicable to solving the rendering equation. First, as the number of samples (trials, runs, tests, etc.) increase, the answer produced by the method becomes closer and closer to the truth. Stated formally: The accuracy of a Monte-Carlo method is directly proportional to the number of samples used.

Secondly, Monte-Carlo methods allow for predictions about a population to be made using samples from that population. The difference between a Monte-Carlo method and any other statistical tool is that the Monte-Carlo method is flexible enough to be applied to a very wide range of situations. While one could simply take the mean of a sample of light rays, the result would be entirely useless. However, using a Monte-Carlo method combined with a sampling distribution proves fruitful.

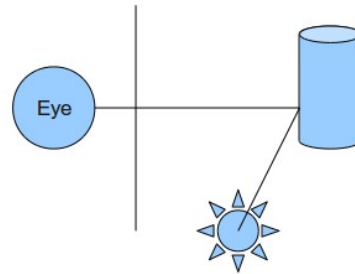
Supercomputing

The complexity (both in terms of sampling and in terms of computation based on those samples) makes Monte-Carlo methods a great candidate for supercomputing. Because these methods require a massive amount of processing power to obtain enough sample data, and because sample data (for the most part) can be taken in parallel, supercomputers seem to be an ideal platform.

Ray Tracing Basics

Description

Ray tracing has been a frequently used solution to the rendering equation. Essentially, a ray tracing model will trace rays out from an eye source, into a scene, to an object in the scene, and then to the light. A ray is traced through every point on the viewing plane (represented in by the vertical line) in this way. Based on various angles of these vectors (and rays, represented by the two-segment line in the image to the right), decisions about shading and location are made. After the tracing process is complete, each point on the viewing plane (which is not actually a line, but a 2D plane) is mapped to a pixel in an image and the color of that pixel is determined.



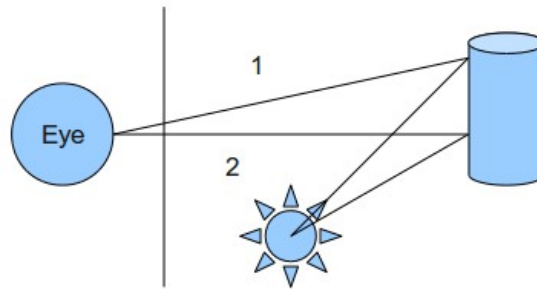
Problems with ray tracing

While ray tracing provides a decent solution to the rendering equation, it comes burdened with several disadvantages. The first and most significant disadvantage is performance. Because there are potentially an infinite number of points on the viewing plane, achieving a perfect render with ray tracing would take an infinite amount of time. Second, ray tracing treats all pixels equally. This means that a pixel where there are clearly no objects (if a ray is traced out from the eye through this point, the ray, upon entering the scene, does not even come close to any objects) is treated in the same manner as a colored pixel. This is incredibly inefficient because an optimized algorithm should concentrate only on important areas of an image.

An overview of shading

One of the difficulties faced by programmers attempting to implement a ray tracing-like model is properly shading each point. Luckily, an observable property of light makes this quite an easy task.

Consider the two traced rays in the image to the right. If one considers what this situation would look like in the real world, one would realize that the bottom of the cylinder would be more brightly illuminated than the top of the cylinder. In this case, the point represented by ray #2



should be brighter than the point represented by ray #1. In order to determine this, consider the normal of the cylinder (a vector coming out directly away from the center column of the cylinder). The angle formed between the normal and the incident ray represents the proportion of shading. The larger the angle, the less illumination. One convenience utilized by many programmers is referred to as “cosine shading” because one could easily use the cosine of the angle between the normal and the incident ray as a coefficient of shading.

3D Calculations

Defining Three-Dimensional Objects

Our three-dimensional world is created using basic shapes, such as spheres, rectangles, cylinders, and cones. We used the basic equations of these shapes combined with parametric equations to define these shapes in our program. For each shape, we need to be able to find three things:

1. Determine if an incoming ray intersects the object and where the collision point is.
2. The reflected or refracted ray from an incoming ray.
3. The cosine of the angle between the incoming ray and the normal vector from the collision point. This value is used to find the amount of lighting that points on the object should receive.

Spheres

A sphere is defined by its center point and radius, and it has the equation

$$(x-c_1)^2+(y-c_2)^2+(z-c_3)^2=s^2$$

with (c_1, c_2, c_3) being the center point and s being the radius. Our incoming ray is defined by a starting point (a, b, c) and slope (p, q, r) . The ray can be represented by parametric equations

$$x=a+pt \quad , \quad y=b+qt \quad , \quad z=c+rt \quad .$$

In order to determine if and where the ray hits the sphere, we need to solve these two equations together. We first plug in the values for x , y , and z from the second equation into the first one.

$$(a+pt-c_1)^2+(b+qt-c_2)^2+(c+rt-c_3)^2=s^2$$

We then expand the expression and write it as a quadratic in terms of t .

$$(p^2 + q^2 + r^2)t^2 + (2ap - 2c_1p + 2bq - 2c_2q + 2cr - 2c_3r)t + (c_1 - a)^2 + (c_2 - b)^2 + (c_3 - c)^2 - s^2 = 0$$

To simplify notation, let

$$l = p^2 + q^2 + r^2, \quad m = 2ap - 2c_1p + 2bq - 2c_2q + 2cr - 2c_3r, \\ n = (c_1 - a)^2 + (c_2 - b)^2 + (c_3 - c)^2 - s^2$$

The previous quadratic equation becomes

$$lt^2 + mt + n = 0$$

To determine the real roots of this equation, we first find the discriminant.

$$d = m^2 - 4ln$$

If this value is negative, then t has no real solutions, and the ray does not hit the sphere. Otherwise, we use the quadratic formula and find the solutions. Plugging these values of t into the parametric equations gives us the points at which the line that contains the ray intersects the sphere. Negative values of t correspond to points behind the starting point of the ray, so they can be eliminated. Then we find the smallest positive value of t , which corresponds to the first point that the ray intersects the sphere, which is the collision point of the ray with the sphere.

To determine the resulting ray from the incident ray, we use the vector component of the incident ray in a formula to produce the vector component of the resulting ray. The starting point of the resulting ray is the collision point. Given incident vector v_i and normal vector n of some surface, the resulting vector is

$$\vec{v}_2 = \vec{v}_1 - 2(\vec{v}_1 \cdot \vec{n}) * \vec{n}$$

The normal to the sphere at the collision point is given by the vector that goes from the center of the sphere to the collision point. We simply plug in the values into the formula and find the resulting vector.

The last part we have to find for sphere is the cosine of the angle between incident vector and the normal. We find this by using the two ways of defining the dot product. Let's call the

incident v , the normal vector n , and the angle between them a . (Note that v_x , v_y , and v_z refer to the x, y, and z components of v .)

$$\vec{v} \cdot \vec{n} = \|\vec{v}\| \|\vec{n}\| \cos(a)$$

$$\vec{v} \cdot \vec{n} = v_x n_x + v_y n_y + v_z n_z$$

Equating these two formulas and solving for $\cos(a)$ gives us

$$\cos(a) = \frac{v_x n_x + v_y n_y + v_z n_z}{\|\vec{v}\| \|\vec{n}\|}$$

Rotation for cylinders and cones

Cylinders and cones are more complicated because the standard equations for them are complicated. However, the standard equations for cylinders and cones that are aligned with an axis are much simpler. To take advantage of this, we define each shape with its parameters, use a rotation matrix to rotate the shape and align it with the z-axis, find the results of the methods, and rotate everything back to its original alignment.

To rotate an object, we first represent the current axis of the object as a vector. (For a cylinder, the axis is the line between centers of the two circles on the ends. For a cone, the axis is the line that connects the point of the cone with the center of the bottom circle). We then find the unit vector that has the same slope as the axis we want to rotate to, which is the z-axis in this case. Using the dot product, we can find the angle between these two vectors. Call this angle a .

Then we use the cross product to find the vector that is perpendicular to the plane containing the first two vectors. This is the axle of rotation about which our object will be rotated. We normalize this vector and call it u . Using the axis-angle formula, the matrix of rotation is:

$$R = \begin{bmatrix} u_x^2 + (1 - u_x^2)c & u_x u_y (1 - c) - u_z s & u_x u_z (1 - c) + u_y s \\ u_x u_y (1 - c) + u_z s & u_y^2 + (1 - u_y^2)c & u_y u_z (1 - c) - u_x s \\ u_x u_z (1 - c) - u_y s & u_y u_z (1 - c) + u_x s & u_z^2 + (1 - u_z^2)c \end{bmatrix}$$

where $c = \cos a$ and $s = \sin a$.

To apply this matrix to object, multiply the vector parameters, like slopes of rays, by the matrix. To adjust points, such as centers of objects, multiply the point by the matrix. But remember that the matrix only rotates objects to align with the z-axis. The axes of the objects do not necessarily coincide with the z-axis. To remedy this, we calculate how far away the axes are from the z-axis along the x and y directions. Then we shift every point by that amount. The result is a transformed object with the same shape, but a different orientation. The axis of the object lies along the z-axis. Afterwards, we shift the points back and multiply the points and vectors by the inverse matrix of the rotation matrix.

Cylinders

Now that we have rotated our objects, cylinders and cones are very similar to spheres and rectangles. The general equation of a cylinder whose axis coincides with the z-axis is given by

$$x^2 + y^2 = r^2 \qquad z_0 < z < z_1$$

where r is the radius and z_0 and z_1 are the top and bottom bounds of the cylinder. Just like before, we define the incoming ray by its starting point (a,b,c) and its slope (p,q,r) . This ray can be written parametrically as

$$x = a + pt \quad , \quad y = b + qt \quad , \quad z = c + rt$$

A ray, if extended, can potentially hit multiple points on the cylinder. Like with the sphere, we find all the points where the ray could hit and take the point that is closest to the starting point of the ray.

First, we check if this ray will ever strike the two faces of the cylinder. We find where the ray would hit the planes of the

faces. Recall that the cylinder is upright and in-line with the z-axis, so the planes of the faces are

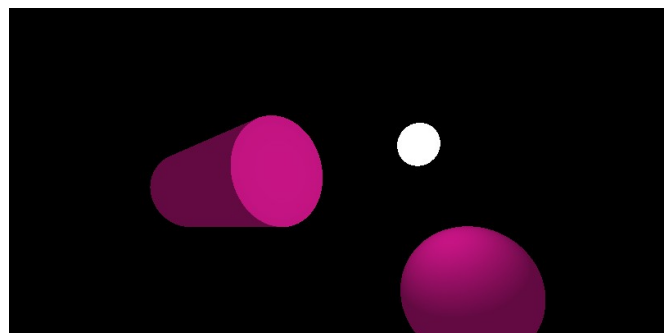


Illustration 1: Cylinders rendered with Metropolis-Hastings

$$z = z_0 \quad \text{and} \quad z = z_1$$

If the ray hits these planes, it will hit the planes when

$$c + rt = z_0 \quad \text{and} \quad c + rt = z_1$$

If the ray does hit one of these planes, we solve for t . We must then determine if the ray strikes the plane inside the face, which is a circle. This is only true if

$$(a + pt)^2 + (b + qt)^2 < r^2$$

If this expression is true, we save this value of t . Then we must determine if the ray strikes the lateral side of the cylinder. We plug the parametric values into the general equation

$$(a + pt)^2 + (b + qt)^2 = r^2$$

We then simplify and write the resulting equation as a quadratic equation with t .

$$(p^2 + q^2)t^2 + (2ap + 2bq)t + (a^2 + b^2 - r^2) = 0$$

The real roots of this equation are values of t (if any) for which the ray hits the lateral side. To reduce the number of calculations that the computer must do, we first find the discriminant, simplify the expression and assign the variable m to this value.

$$m = 4(b^2 p^2 - a^2 q^2) + 4(r^2 p^2 + r^2 q^2)$$

If the discriminant is negative, the equation has no real roots. Otherwise, we use the quadratic equation to find the roots.

$$t = \frac{-2ap - 2bq \pm \sqrt{m}}{2(p^2 + q^2)}$$

Out of all the values of t we have so far, we find the first place that the ray hits the cylinder, which corresponds to the smallest value of t . However, negative values of t correspond to points that are behind the starting point, so we must find the smallest positive value. We plug this value of t back into our parametric equation to find the point that the ray hits the cylinder.

We find the resulting ray with the same formula as before. To find the normal vector of

the cylinder at the collision point, we rotate and shift the cylinder to align with the z-axis and determine whether the collision point is on the two faces of cylinder or on the lateral surface. If the point is on the top face, the normal is the vector in the positive z direction. If the point is on the bottom, the normal vector is in the negative z direction. If the point is on the lateral surface, the normal is the vector parallel to the xy plane that intersects the axis and the point. For example, if the the point is (x, y, z) , the normal is the vector $(x, y, 0)$. We then plug this value and the incident vector into our resulting vector formula.

Finding the cosine of the angle between the incoming ray and the normal is simple. We have already found the normal, and we just use the definitions of the dot product like before.

Cones

After rotating and shifting the cone to align with the z-axis, the general equation is

$$\frac{x^2 + y^2}{m^2} = (z_0 - z)^2 \quad \text{where} \quad m = \frac{\text{radius}}{\text{height}}$$

Just like with the other shapes, we check first if the ray hits the bottom circle. Then we write the ray as a parametric equation, plug the values into the general equation, write the expression as a quadratic equation, and find the real roots. We take the smallest positive value of t and plug this value back into the parametric equations to find the collision point.

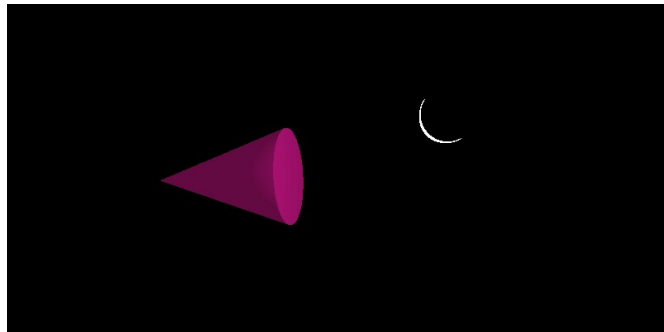
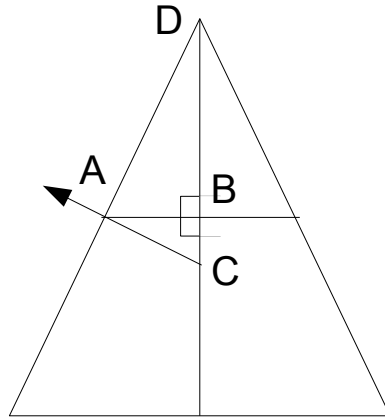


Illustration 2: A cone rendered by Metropolis-Hasting

Finding the resulting ray is a little more difficult that before because finding the normal vector is more complicated. If the point is on the bottom circle (or the top circle, if the cone is inverted), the normal is either the negative z-direction or the positive z-direction. For points on the lateral surface, we've provided a figure on the next page shows a cross section of the cone. Point A (x, y, z) is the collision point, point B $(0,$

$0, z)$ is the point on the axis of the cone that is on the same horizontal plane as A. The vector between C and A is the vector we are trying to find. Let's call this vector n .



We note that

$$\Delta ABC \sim \Delta DAB$$

They both have a right angles, and

$$\angle CAB + \angle ACB = 90^\circ = \angle ACB + \angle CDA$$

So, $\angle CAB = \angle CDA$ and the triangles are similar. We can use the dot product definitions to find the angle between the axis and the vector between D and A. This is $\angle CDA$, which is equivalent to $\angle CAB$. Let's call this angle a . By the definition of sine,

$$\sin(a) = \frac{\overline{CB}}{\overline{AB}}$$

We solve for \overline{CB} . Since we know the coordinates of B, the distance between B and C, and the fact that C is also on the z-axis, we can find the coordinates of C. From the coordinates of A, B, and C, we can find the vector from C to B and the vector from B to A. The vector sum of these vectors is the vector between C and A, which is what we want. We plug this vector and the incident vector in our resulting vector formula to find the resulting ray.

Since we now know the normal vector on the cone from the collision point, we can use our dot product definitions to find the cosine of the angle between the incident vector and the normal vector.

The Metropolis-Hastings Light Transport Algorithm

Introduction and disclaimer

The Metropolis-Hastings algorithm is a specific Monte-Carlo method that approximates the distribution of functions that can not be directly sampled. In other words, the Metropolis-Hastings algorithm lets one use sample data to approximate the distribution of the entire population.

In terms of rendering, the Metropolis-Hastings algorithm requires a slight deviation from the formal statement of the algorithm. Therefore, the methods documented here should not necessarily be considered an implementation of the Metropolis-Hastings algorithm, but should instead be considered as a Monte-Carlo approach to rendering rooted in the logic of the Metropolis-Hasting algorithm.

Explanation of the algorithm

The algorithm begins by tracing a finite number of rays from the camera/eye through evenly spaced points on the viewing plane into the scene. These rays are then stored in memory as three points: the point where the ray intersects the viewing plane (this point will be referred to as P), the point where the ray strikes an object in the scene (S), and the point where the ray hits the light source (L). It is important to note that an image can be generated from this point, and a sample image is shown in Illustration 3. Notice the black lines and dots present from a lack of sampling in certain regions, as well as the “hard” shadows present behind both spheres.

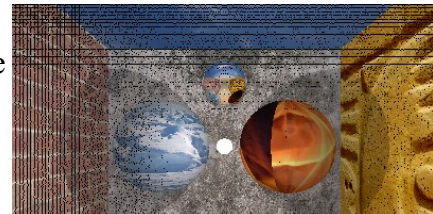


Illustration 3: An image generated from only ray tracing

Let the overall population of light rays represent every single possible light ray traveling from the light source to the eye/camera. The list of rays generated from the first stage of our algorithm serves as a random sample of this population. Because the population size is virtually infinite (as there can be an infinite number of modeled rays from the light source to the eye), the

distribution of the list of rays taken will be approximately normal, as per the central limit theorem. With this known, the algorithm creates several distributions based upon rays that pass through the viewing plane in close proximity to each other. Each of these groups of rays form a “ray cluster,” a group of rays that have similar properties because they are within close proximity within the image. Realistically, there will probably be between 1,000 and 100,000 ray clusters for a given image.

Each ray cluster forms nine graphs/distributions – one for each coordinate in each point of each ray. For example, the P point's X, Y, and Z components are each plotted in their own distribution. The algorithm calculates the mean and standard deviation (which, in this case, is actually the standard error) of each distribution and uses the results as parameters to create a normal sampling distribution. The algorithm then takes random samples from each of these distributions to propose a new ray. This proposed ray is comprised of 9 random points taken from 9 different sampling distributions. Thus, a proposed ray can be defined as:

$$P\{X, Y, Z\} S\{A, B, C\} L\{Q, R, S\}$$

Once this proposed ray has been generated, it is tested against the scene itself (i.e., the algorithm makes sure that the proposed ray is actually valid within the modeled world). If the proposed ray is added, the proposed ray is added to the current ray cluster and the mean and standard deviation of the normal distribution being used is recalculated and another ray is proposed. This process repeats until an adequate number of rays have been calculated, or a finite time has been reached. Illustration 4 is what

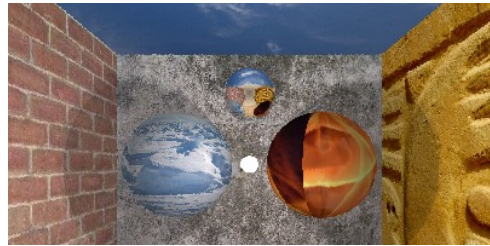
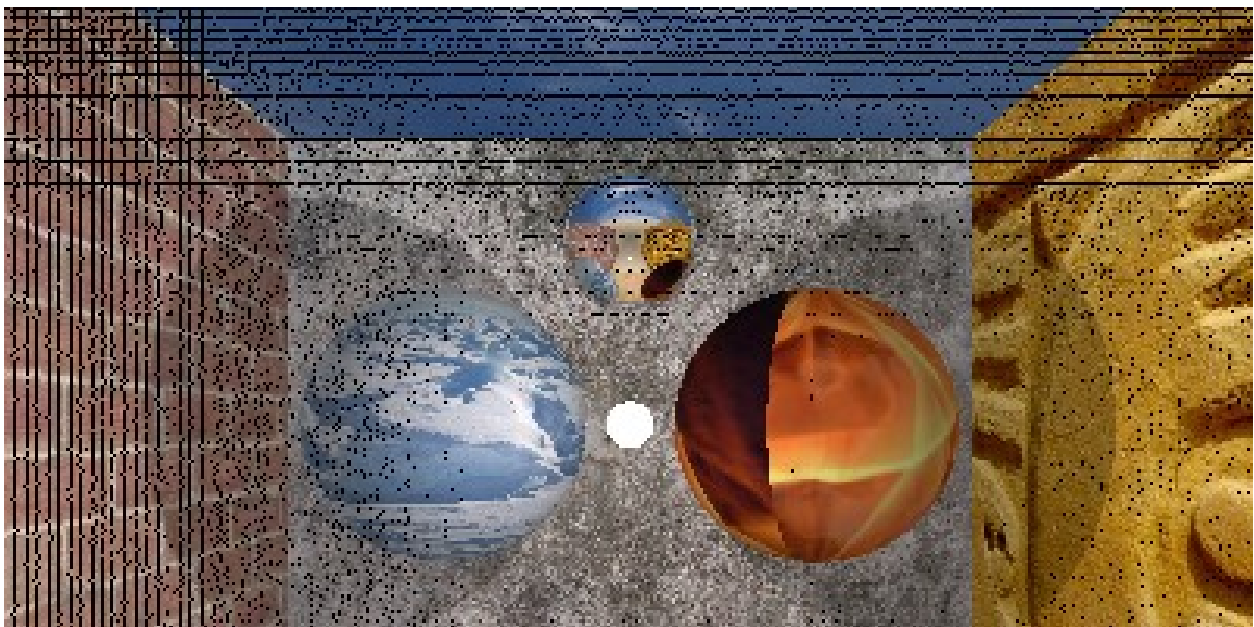
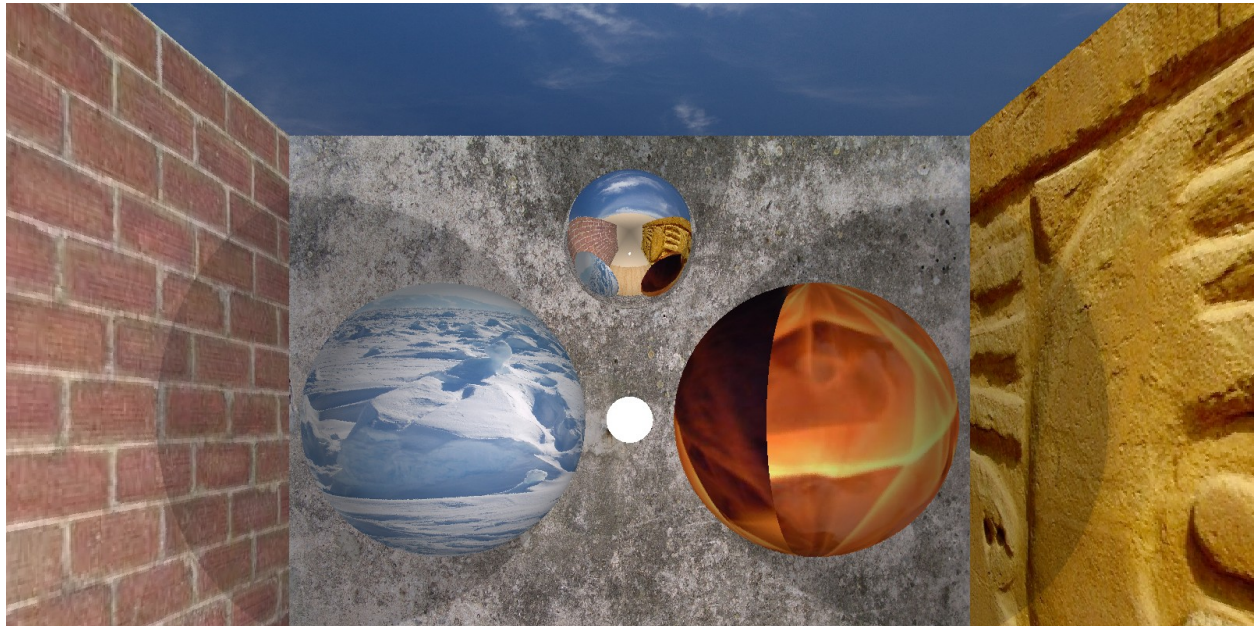


Illustration 4: Low quality Metropolis-Hasting render

Illustration 3 looks like after the Metropolis-Hasting's algorithm finds 1280000 rays. The result speaks for itself.

Image comparison

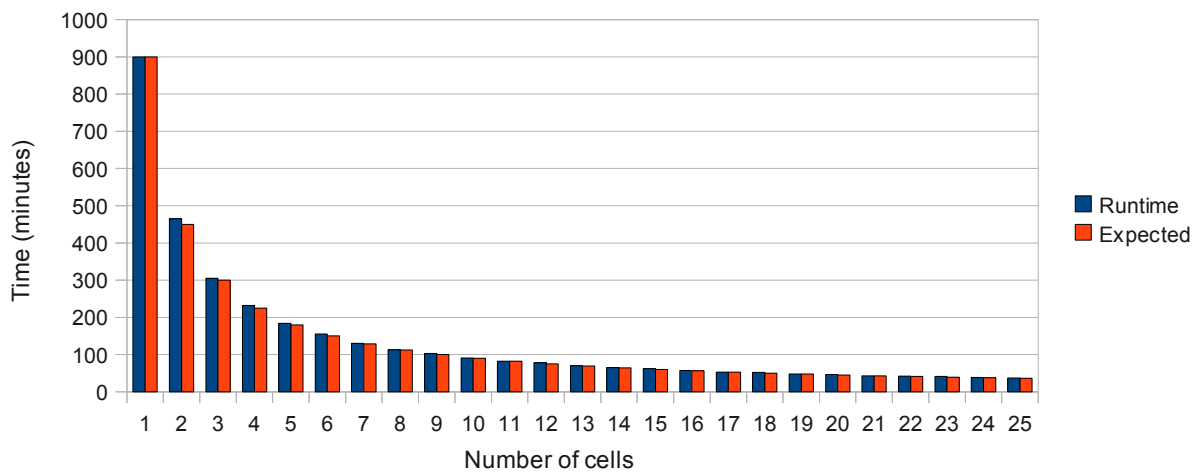
The first image is a high quality image produced by the Metropolis-Hastings algorithm. The second image is created from standard ray tracing. Specifically, note the softer shadows and lack of graininess in the Metropolis-Hastings image.



Parallel advantages of the algorithm

Rendering algorithms are great candidates for parallel computing. In fact, most modern day graphics cards (such as those designed by nVidia and ATI) contain hundreds of vector processors for quickly rendering images. Sadly, the methods and algorithms used are aimed at creating realistic *looking* graphics, not an accurate model of light. However, the same capabilities for parallelization exist within the algorithm's more accurate and realistic model of light.

Scaling factors



The graph above depicts the runtime for a high-quality render using the Metropolis-Hastings algorithm. The blue bar represents the time it took for a given number of cells (in this case, each cell is a separate computer running 3 threads) to complete the render. The red bar represents how long it would take to render the image in a perfect world – i.e., one with perfect scaling. For example, with one cell the render took 900 minutes. In a perfect world, eight computers would be able to perform the same task eight times faster, which would be 112.5 minutes. The algorithm took 113 minutes with 8 cells, demonstrating that the algorithm is incredibly distributable, and thus an excellent candidate for a supercomputer.

There are three properties of the algorithm and its implementation that make it particularly distributable:

1. Each ray cluster can be calculated independently of other ray clusters. This has

two implications. First, it means that each cell requires no communication with the cells around it. Each cell only needs to be able to talk to one central cell (perhaps called a server). Second, each ray cluster can be calculated at the same time, which means as the number of cells increase, the algorithm continues to scale because the number of clusters will always exceed the number of cells (one can control the number of ray clusters for any given image, giving the algorithm almost infinite scalability).

2. The actual rendering – the creation of the pixels themselves – can be done independently from each ray cluster. This means that each cell does not have to send back all the data contained in its ray cluster, only the RGB values (red, green, and blue) of each pixel the cell was assigned to calculate. This provides for a very low amount of network traffic, thus decreasing the cost of adding an additional cell.
3. The algorithm has a low traffic-to-processing ratio. This means that very little data needs to be sent across the network to instruct each cell of what it needs to do. Each cell needs only two components in order to operate: the area of the viewing plane that contains the ray cluster a given cell would be responsible for, and a copy of the scene. The area is only two doubles, one representing a starting point and another representing a stopping point. The digital world is small as well, and could even be preloaded. One may think the scene would take up a lot of space, but keep in mind that a sphere is really just a point and a radius. Everything within the algorithm is represented in terms of dimensions.

Video produced by the algorithm

Little known fact: The movie Shrek took several years to render. In designing the algorithm, it was pointed out that if high quality images could be produced, then high quality video could also be produced by stitching several images together. The results proved illuminating (pun intended). Because this medium (paper) does not easily allow for the

publishing of a video file, a sample video produced by the algorithm can be viewed here: <http://marcusfamily.info/~ryan/Export.mov>

The ability to render video makes the algorithm an even better candidate for a supercomputer. Because each frame can be rendered independently of each other, the algorithm can efficiently encompass even more cells while gaining a phenomenal performance boost.

Conclusions

While the algorithm described here (and developed, in full, by team 69) does not come close to providing an all encompassing model for light, it does succeed in providing a supercomputer-ready, high quality, and incredibly accurate model of light.

Through a combination of statistics, theory, and the power of computers, this implementation of the Metropolis-Hastings Light Transport algorithm was not only incredibly successful, but also a pleasure to develop.

Light is a tricky thing. It surrounds every human everyday, and is probably one of the most taken-for-granted elements of human life. Hopefully, this algorithm will provide some insight into the surprisingly dark mystery of light. That is everyone's goal anyway: enlightenment.

A wise grandmother once said: “computers seem to be a compilation of pretty pictures on top of loud boxes.” The algorithm creates some of the prettiest pictures on some of the loudest boxes, so at a minimum, it is grandmother-approved.

PARTICLE-ATMOSPHERE INTERACTION¹

NM Supercomputing Challenge
Final Report, March 23, 2010

Team 84 McCurdy High

Team Members:

Dennis Trujillo
Oliver Galvan
Brandon Ricci

Project Mentors:

John Pretz
Brenda Dingus
Philip Sanchez

¹ Amended from 'Gamma-Muonic Flux Supernovae Correlation Model'

1. Table of Contents	2
2. Executive Summary	4
3. Introduction.....	5
3.1 Problem Statement	6
3.2 Procedure Overview	6
4. Background Research	7
4.1 General Particle Information.....	7
4.2 Equations Used.....	8
4.3 Air Shower Evaluation.....	10
4.4 Hadronic VS. Gamma Particles	11
5. Procedure: Particle Determination.....	13
5.1 CORSIKA Simulation.....	13
5.2 Physical Model.....	13
6. Analysis of Methods	17
7. Sample Results	18
7.1 Sample Data.....	18
7.2 Data from Distcalc.cpp.....	18
7.3 Data from PID-Distributions.cpp.....	19
7.4 Daughter Particle Spread.....	19
7.5 Particle Distance from Center.....	21
7.6 Proton Shower Images.....	24
7.7 Gamma Shower Images.....	25
8. Future Work	27

9. Conclusion	28
10. Acknowledgments	29
11. Bibliography.....	30
11.1 Internet Resources.....	30
12. Appendix	31
12.1 Particle-Identification Table.....	31
12.2 Data Read Code.....	33
12.3 Particle Determination Code.....	46

2. Executive Summary

As a result of cosmic radiation interaction with the atmosphere we find that a series of phenomena known as extensive air showers results from the decay of these particles when they make contact with the gases and other elements our atmosphere is composed of. Because each particle decays differently depending on the strength of the forces that bind it we find that air showers produced by different particles produce a different array of daughter particles which if unstable continually break up into other subatomic particles. Because of such we can determine what kind of particle each air shower is produced by based on the pattern of decay left as it approaches ground level and the energy of the particle causing the air shower.

The purpose of our project is to determine the difference between gamma and electromagnetic air showers. We are going to solve this problem with the help of the CORSIKA computational model. The CORSIKA model itself is a Monte Carlo system, that is to say, that it randomly chooses numbers for its plotting. We want to evaluate the distribution of particles at seven thousand feet and first interaction relative to density that the GNUplots produce. All graphs are plotted on a Cartesian coordinate system with the center of the interaction as the origin. If a particle is a proton there will be much more energy towards the center of the interaction. If a particle is a gamma then it will not have as much particles and they will be more spread out towards the center of the interaction. Another means of determining the differences is by evaluating the different daughter particles that each interaction produces. For example if a particle is a proton it will produce muons. If it the particle is gamma then it will produce solely electromagnetic daughter particles. We are not going to launch these results once, but more like a thousand times in order to gain a more accurate understanding of the characteristics associated with air shower events.

3. Introduction

Extensive air shower simulation is an area of study which has great potential relative to the understanding of particle decay and development as evidenced by findings related to particle decay and daughter particle development as a result of air showers resulting from interactions similar to those evaluated within our study. Such studies can result in information relative to the stability of certain particles, the decay rate of these particles, and the creation of daughter particles. Essentially the evaluation of particle interaction within nature is comparable to the work done at CERN with the LHC although with real interactions being taken into consideration. Therefore we find that the study and simulation of extensive air showers is a vital and cheap means of evaluating particle characteristics and the forces which drive the universe as a whole.

The primary goal of several institutions around the country and throughout the world is particle characterization as a result of computer simulation and physical experimentation in order to produce a more full understanding of the components which account for all matter and likewise hold the key to understanding why the universe exists in the means in which it does. These simulations and experiments are integral to one another; each exists to prove or test the other. We find that code can be written according to theory but the true test of its worth can only be found through experimentation, and similarly experiments need to be designed according to certain standards which can be determined through computer simulation.

A companion approach involving the use of both physical experimentation and computer simulation can provide the best means of evaluation. A large scale analysis of data produced by proven code and the implementation of physical experimentation could potentially lead to the association of certain before unknown characteristics with certain particles, a cheap means of particle study yielding the same results as something as complex as the LHC, and potentially provide information relative to why the universe exists as it does and perhaps lead to the discovery of before unknown particles.

3.1 Problem Statement

How can raw data be accurately used to potentially identify the parent particle causing an extensive air shower event, i.e. Hadronic vs. electromagnetic in origin? How can these observations be used to further our knowledge of particle behavior and characteristics?

3.2 Procedure Overview

Our study begins with the acquisition of raw data produced by a series of simulations validated by code for data analysis. Such data is produced by the CORSIKA version 6900 program, a Monte Carlo simulation package which simulates the interaction and break up of certain particles and light nuclei. This data contains a series of parameters relative to particle decay and interaction, including parent particle type, daughter particle types, and energy at ground level interactions of 7000 ft in elevation.

Using this data we then compare the number of particles produced, the types of daughter particles, and energies at ground level to the parent of the original interaction. These are then plotted on a curve in order to determine the fundamental differences between particle interactions and establish traits common to each parent as it decays.

This means of distinguishing particles is commonly used and can potentially be used to classify and determine before unknown characteristics of particles as they decay and interact. We have meshed the CORSIKA version 6900 software with our own coding in order to write a program which can take raw data from any source, in terms of daughter particle types, total energy in GeV or TeV, and number of particles produced and provide a fairly guess as to what particle initiated the interaction in question.

4. Background Research

4.1 General Particle Information

Particles responsible for extensive air showers must have a source. Stars, supernova, black holes, and similar objects produce radiation. This radiation comes in the form of particles. Different particles decay faster than others depending on how stable they are. All particles decay at different rates depending on how stable they are and the rate of velocity with which they collide with an object or material in space. All particles have different charges and masses. Relative to our project we are looking at protons, muons, and gamma particles. For example less stable particles such as pions and kaons produce a cascade faster as a result of but they are not relative to our project.

A cascade occurs when particles decay producing more and more particles; a chain reaction occurs. The explosion produces masses of particles. Cascades produce particles exponentially. These cascades occur 24 hours a day 7 days a week. Cascade simulation is important because it is the main focus of our project and it provides a means of studying particle interaction cheaply.

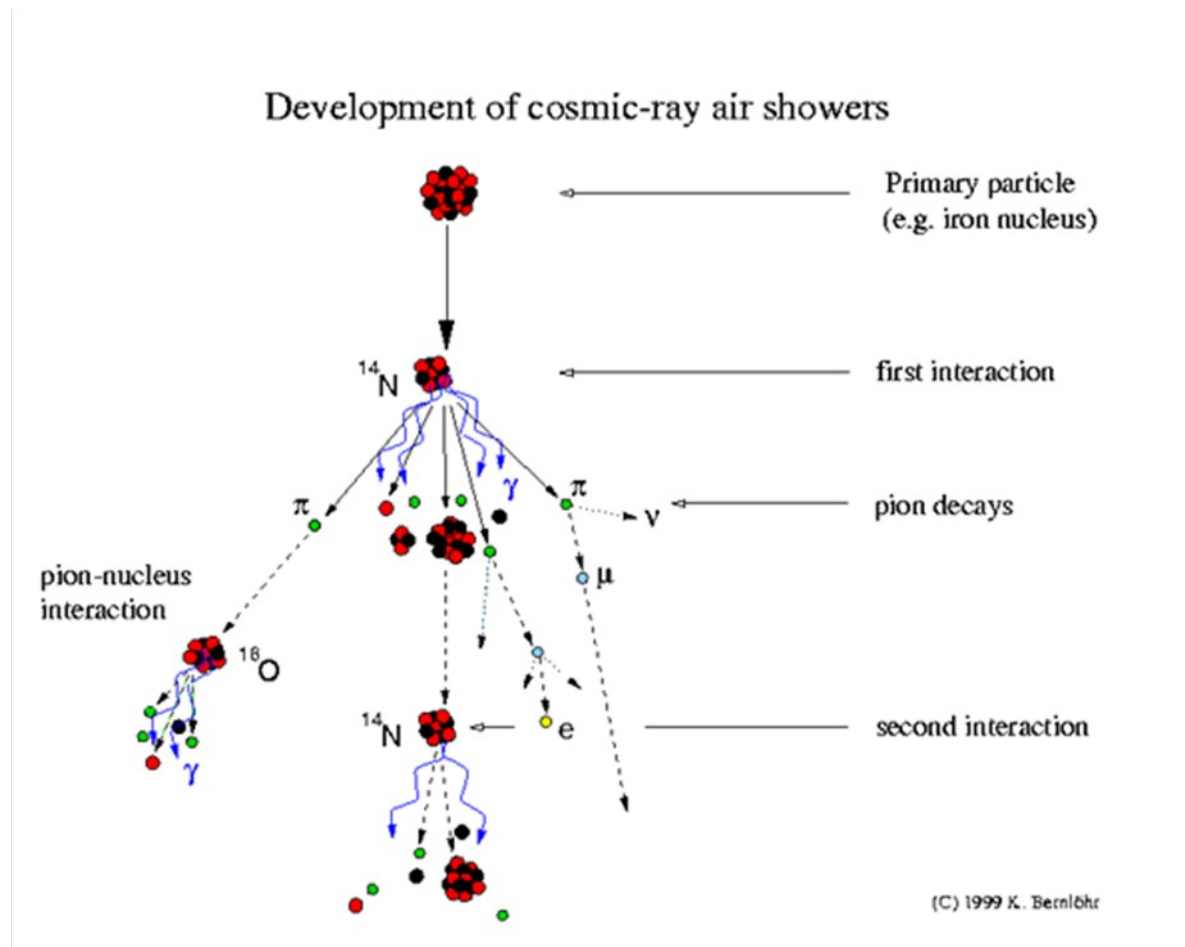
Since protons have a charge and a mass they are affected by the gravitational fields of planets, supernovas, black holes and objects in space of which either hold a strong enough gravitational pull or magnetic field to interact with the particle's trajectory. These influences alter the path of the particles when they reach the earth. Particles will decay if they encounter an object at a high enough velocity no matter how stable a particle is. This includes protons. Even though they are very stable they will still decay. We are aware that the magnetosphere and material interaction does play a role in our CORSIKA program, and have set out to identify and understand the corresponding equations. When protons hit the magnetosphere, the proton particles will be repelled, since both protons and the magnetosphere have different charges. If the proton has a high enough velocity it will break up and decay through. When they hit the earth the cascade the parent particle produces can be seen as

containing several characteristics common of the parent which produced it.

Gammas are the opposite from protons; they do not have a charge and a mass. So they are not affected by gravitational pulls of objects in space. They will also produce daughter particles but that is unlikely.

We are going to gather the attributes of both Hadronic and electromagnetic air shower interactions and with the help of our computational model determine the differences between these two groups of particle interactions.

2



4.2 Equations Used

The following equations are integral to calculating the outcome of interactions between particles

2 <http://lpsc.in2p3.fr/TPsubat/m2/cosmic-rays.jpg>

and a material and are essential to understanding the interaction between the atmosphere and a particle. These equations describe the phenomena accurately and are essential to describing several particle characteristics.

Bethe-Block Stopping Formula

$$dE/dx = \frac{\lambda z^2}{\beta} \kappa_1 (\ln(\gamma - 1) - \beta + \kappa_2) = \frac{\lambda \gamma^2}{\gamma^2 - 1} \kappa_1 (\ln(\gamma^2 - 1) - \beta + \kappa_2)$$

$\beta = v/c$, γ is its Lorentz factor, z is the charge of the ionizing particle in units of e . $\kappa_1 = 0.153287 \text{ MeV g}^{-1} \text{ cm}^2$ and $\kappa_2 = 9.386417$ are derived from values for dry air

Deflection in Earth's Magnetic Field

$$\alpha \approx lZ \frac{[\vec{p} \times \vec{B}]}{p^2} \text{ Hadronic vs}$$

This is a description of particle deflection in Earth's magnetic field, where l = length, z = charge, \vec{B} vector = magnetic field vector, \vec{p} vector = particle momentum

Time of Flight Hadronic vs

$$dt = \frac{l}{c \beta_{ave}}$$

At the first interaction of the primary in the atmosphere, the timing of the shower is started. The time interval dt is the time elapsed as the particle moves along its path; dt is calculated by dividing the path length l by the average particle velocity, where β_{ave} is the arithmetic mean of the particle at the beginning and end of the trajectory.

Mean Path

$$\lambda_{tni} = m_{air} / \sigma_{tni}$$

Describes the interaction between muons and a material, in this case air. The mean free path for these interactions is given by the equation above where $m_{\text{air}} = 14.54$ is the average atomic weight of air in g/mol and λ_{tni} is given in g/cm^2 .

Probability of Material Traverse

$$P_{\text{tni}}(\lambda) = \frac{1}{\lambda_{\text{tni}}} e^{-\lambda/\lambda_{\text{tni}}}$$

Describes the probability of a muon to traverse an atmospheric layer of thickness λ without corresponding interaction.

4.3 Air Shower Evaluation

We find air showers to be reliant on the particles that produce them; the particle that produces an interaction will inevitably affect what happens during this process. Therefore we find that an interaction which contains a gamma as a parent will decay differently than a proton or other particle, and as a result produce a different numbers of daughter particles as opposed to a proton shower produced under the same conditions. Likewise these particles will differ in terms of final ground level energy, and daughter particle types.

Quarks	2.4 MeV $\frac{2}{3}$ $\frac{1}{2}$ u up	1.27 GeV $\frac{2}{3}$ $\frac{1}{2}$ c charm	171.2 GeV $\frac{2}{3}$ $\frac{1}{2}$ t top	0 0 1 γ photon
	4.8 MeV $-\frac{1}{3}$ $\frac{1}{2}$ d down	104 MeV $-\frac{1}{3}$ $\frac{1}{2}$ s strange	4.2 GeV $-\frac{1}{3}$ $\frac{1}{2}$ b bottom	0 0 1 g gluon
	<2.2 eV 0 $\frac{1}{2}$ ν_e electron neutrino	<0.17 MeV 0 $\frac{1}{2}$ ν_μ muon neutrino	<15.5 MeV 0 $\frac{1}{2}$ ν_τ tau neutrino	91.2 GeV 0 1 Z⁰ weak force
Leptons	0.511 MeV -1 $\frac{1}{2}$ e electron	105.7 MeV -1 $\frac{1}{2}$ μ muon	1.777 GeV -1 $\frac{1}{2}$ τ tau	80.4 GeV ± 1 1 W[±] weak force
				Bosons (Forces)

4.4 Hadronic VS. Gamma Particles

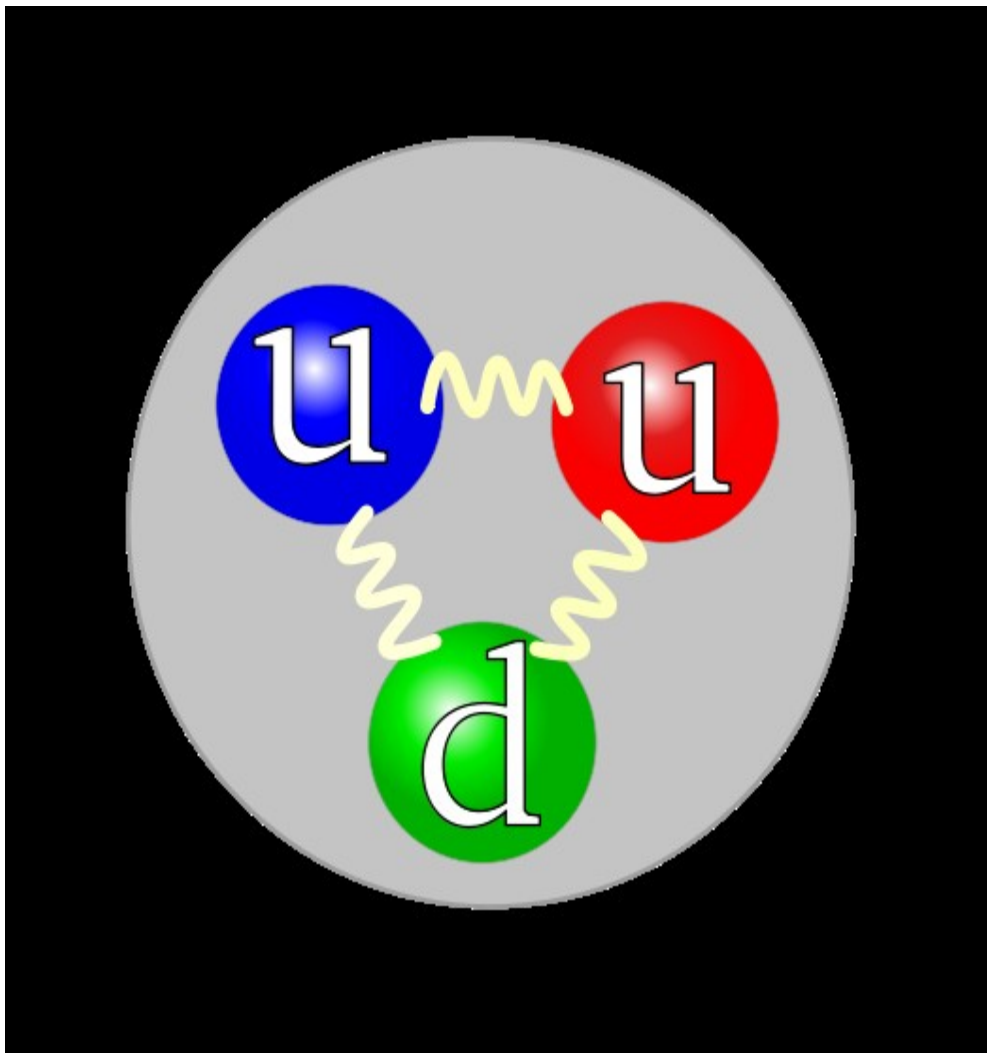
In terms of our project we are solely identifying the differences between Hadronic air shower events and electromagnetic or gamma induced events. These particles have a series of major fundamental differences in terms of the physical qualities of the individual particles themselves and in terms of the air showers they produce, as evidenced by our work. Some of the fundamental differences between protons, the parent of Hadronic showers, and gamma particles can be found in variances in charge, i.e. protons have a charge of $1+e$, mass of $1.672621637(83) \times 10^{-27}$ and a mean lifetime of $>2.1 \times 10^{29}$ yr. Gamma particles however have no mass, no charge, and are stable with an indefinite lifetime (see figure above). Likewise we find that air shower events produced by differing particles

3 Standard Model of Particle Physics

decay differently as a result of their stability and makeup.

⁴Model of Proton

We find protons to be composed of one down and two up quarks as evidenced by the model below.



4 http://en.wikipedia.org/wiki/File:Quark_structure_proton.svg

5. Procedures

5.1 CORSIKA Simulation

The computational model we are implementing in our project is written in the FORTRAN and C programming languages and is known as CORSIKA version 6900. CORSIKA is a Monte Carlo program, meaning it sets random values for simulation, hence it looks at a wide array of simulations and interactions at different energies and altitudes. Because of such we find that CORSIKA is non-limiting and useful for a wide variety of terms. Besides imaging gamma and proton showers, which are the focus of our project, CORSIKA also simulates air showers created by other subatomic particles, nuclei of certain elements, and photons.

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A
 000 000 0000 0000 00 0 0 0
0 0 0 0 0 0 0 0 00 0 0 0 0
0 0 0 0 0 0 0 0 00 0 0 0 0
0 0 0 0 0 0 0000 00 00 0 0
0 0 0 0 0000 0 00 0 0 0000000
0 0 0 0 0 0 0 0 00 0 0 0 0
 000 000 0 0 0000 00 0 0 0 0

COSMIC RAY SIMULATION FOR KASCADE

A PROGRAM TO SIMULATE EXTENSIVE AIR SHOWERS IN ATMOSPHERE

BASED ON A PROGRAM OF P.K.F. GRIEDER, UNIVERSITY BERN, SWITZERLAND
QGSJET MODEL ACCORDING TO N.N. KALMYKOV AND S.S. OSTAPCHENKO, MSU, MOSCOW, RUSS
IA
HDPM MODEL ACCORDING TO J.N. CAPDEVIELLE, COLLEGE DE FRANCE, PARIS, FRANCE
GHEISHA ROUTINES ACCORDING TO H. FESEFELDT, RWTH AACHEN, GERMANY
EGS4 ACCORDING TO W.R. NELSON, H. HIRAYAMA, D.W.O. ROGERS, SLAC, STANFORD, USA
NKG FORMULAS FOR FAST SIMULATION OF EL.MAG. PARTICLES

REFERENCES: D. HECK, J.KNAPP, J.N. CAPDEVIELLE, G. SCHATZ, T. THOUW,
REPORT FZKA 6019 (1998)
SEE ALSO WEB PAGE http://www-ik.fzk.de/corsika/

INSTITUT FUER KERNPHYSIK
FORSCHUNGSZENTRUM KARLSRUHE
POSTFACH 3640
D-76021 KARLSRUHE
GERMANY

IN CASE OF PROBLEMS CONTACT: Dr. Tanguy Pierog
e-mail: tanguy.pierog@ik.fzk.de
FAX: (49) 7247-82-4075
PHONE: (49) 7247-82-8134
OR : Dr. Dieter Heck
e-mail: dieter.heck@ik.fzk.de
FAX: (49) 7247-82-4075
PHONE: (49) 7247-82-3777

AND SEND YOUR LIST-FILE BY E-MAIL

NUMBER OF VERSION : 6.900
5
```

5.2 Physical Model

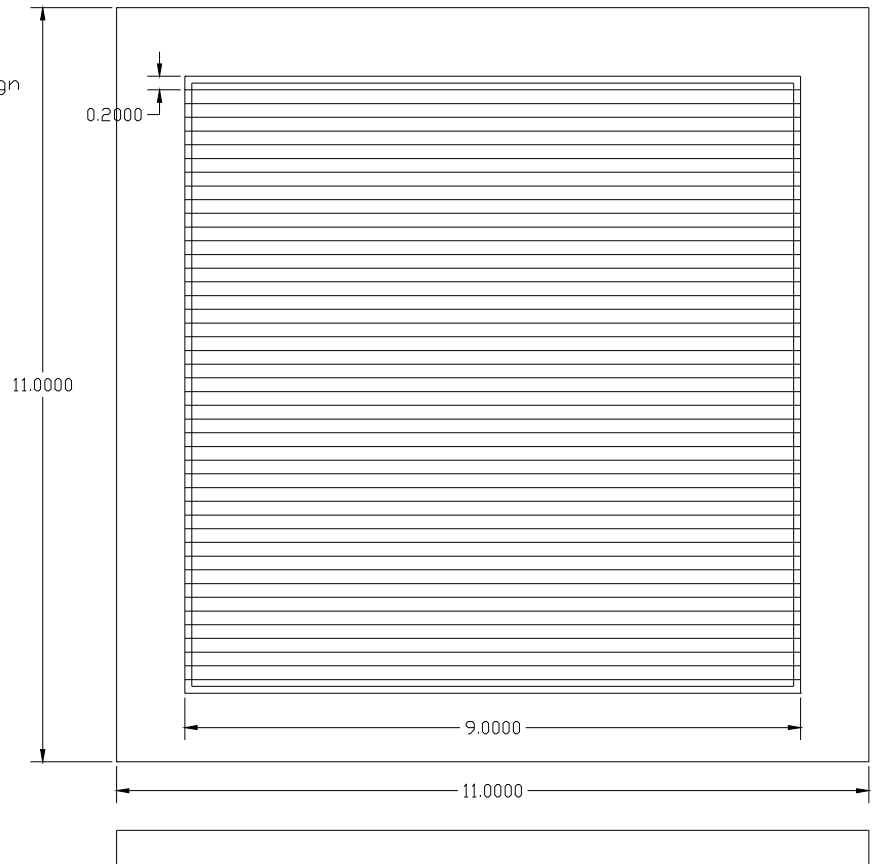
Our physical model is based on a wire array detector known as a Geiger Muller detector and consists of several arrays of thin wire, approximately 250 micrometers in diameter, which is laid out

and charged with high voltage. These detectors are assembled in an array and are designed to have several layers of wire arrays. The physical properties of these detectors allow particles with a charge to be counted because when these particles encounter the array they interact with the voltage carried through the sense wires and change the overall voltage running through the system, which initiates our counter to consider this small change as a particle interaction. We must allow for arrays to be layered in order to effectively determine a particle's origin. The reason for such lies in the fact that background radiation interacts with the detector and can cause changes in voltage within one array although if data from layered arrays is considered we can consider background radiation ruled out. Such phenomenon exists because we find that background radiation has a low implicit velocity and charge, therefore it should cause a change in only one array although we find that particles resulting from cosmic rays and their daughter particles hold an inherit high charge and velocity and should likewise pass through multiple detectors easily. With the combination of a semi large array of particles we should be able to determine the pattern produced by several real air showers and compare them to data found through our simulations in CORSIKA and our original coding to determine the parent particle which initiated the shower.

At this point we have not been able to run our detector due to problems associated with the particle counter and power supply. This side project is still in progress and we intend to complete and run it in order to possibly compare data collected from this with our own coding and the CORSIKA models.

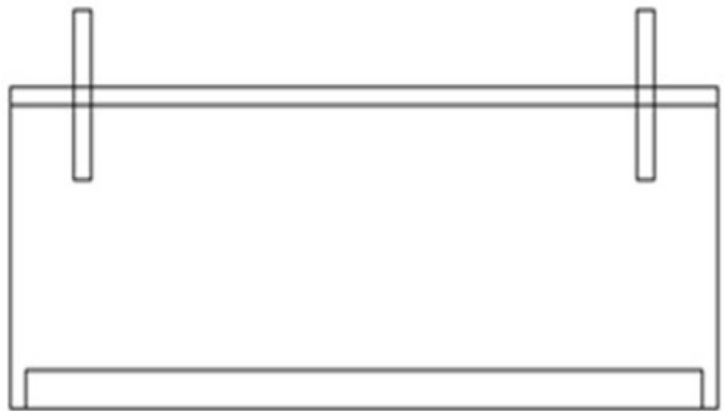
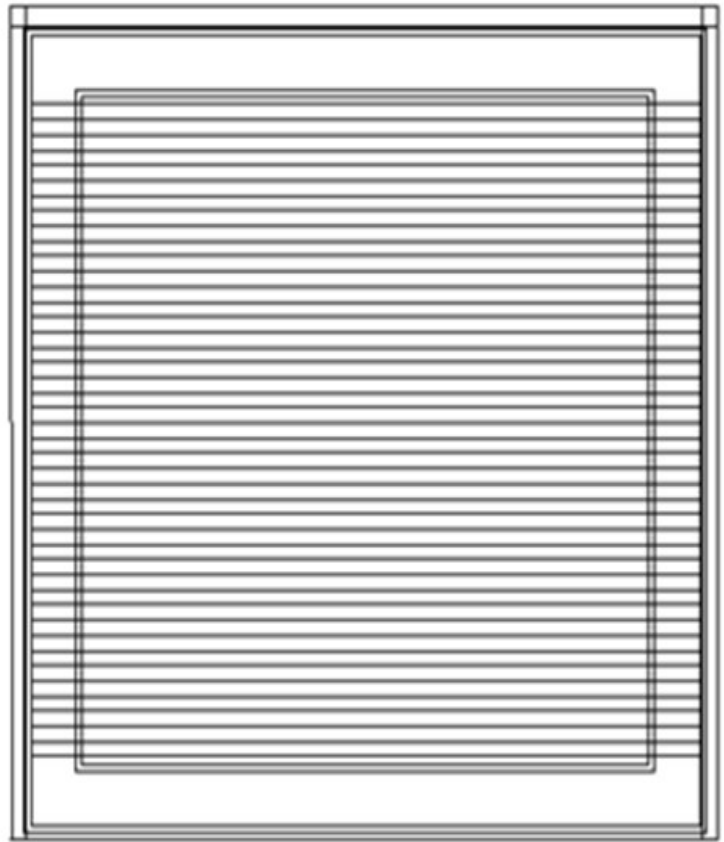
Particle Detector: Wire Array Design

wire thickness: 250 micrometer



6 Original particle detector ddesign

Array Housing-Particle Detector
Array



6. Analysis of Methods

One of the methods we are going to use involves the evaluation the series of plots and graphs that were produced by CORSIKA and our own coding. We are going to use these graphs as a means of determining the differences between Hadronic and Electromagnetic air showers. Through our research we have discovered that the methods we have employed, evaluating daughter particle distances from the center, and evaluating daughter particle types produced are effective ways of determining the differences of Hadronic and Electromagnetic air showers.

In one of our graphs we are comparing the distance of daughter particles from the center of the interaction from both proton and gamma showers. We intend to determine similar qualities between similar particles in order to produce a means of particle identification.

In another one of our graphs we are comparing the daughter particle spread of both Hadronic and Electromagnetic showers. This method helped us determine the different types of daughter particles that will be produced between both Hadronic and Electromagnetic showers in one interaction.

In our plots we are comparing the images of the Hadronic and Electromagnetic showers. Some visual differences we need to analysis in the plots are high amounts of energy or low amounts of energy, if the energy is more compressed or more spread out, and whether or not the interaction produces daughter particles or not.

If the air shower is produce by particles that have a mass and a charge then visually the plot will reveal high amounts of energy, the energy will be more compressed, and there will daughter particles like muons present. If the air shower is produced by particles without a mass and a charge then there will be lower amounts of energy, the energy will be more spread out, and there will be no daughter particles.

7. Sample Results

7.1 Sample Data

The following data has been produced as a result of our own coding and the COSIKA software and represents a fragment of the data used for our study.

7.2 Data from Distcalc.cpp

The following data excerpt represents particle distance versus bin number produced by the Distcalc.cpp program. This data is visualized in the 'Particle Distance from Center' plots following.

0 2670
1 2527
2 2054
3 1710
4 1363
5 1060
6 898
7 740
8 550
9 498
10 437
11 377
12 294
13 276
14 244
15 207

16 188
17 165
18 153
19 129
20 132
21 108
22 86
23 96
24 73
25 74
26 73
27 66
28 71
29 43
30 50
31 55
32 50
33 38
34 46
35 41
36 25
37 31
38 37
39 22

7.3 Data from PID-Distributions.cpp

The following data represents the number of daughter particles versus daughter particle types produced by our PID-Distributions.cpp program, and is visualized directly below in the 'Daughter Particle Spread' plots.

Gamma Parent

0 637

1 430627

2 18410

3 33565

5 55

6 61

Proton Parent

0 366

1 755891

2 37362

3 64554

5 7493

6 7302

7.4 Daughter Particle Spread

The following graph represents daughter particle types produced by a pool of gamma and proton interactions, focusing on particle types 1-6. We find that for the given data set that there is a subtle

although noticeable difference between the daughter particles produced by both gamma and proton parents between the number of type five and six particles. This difference therefore provides a possible means of particle identification and differentiation.

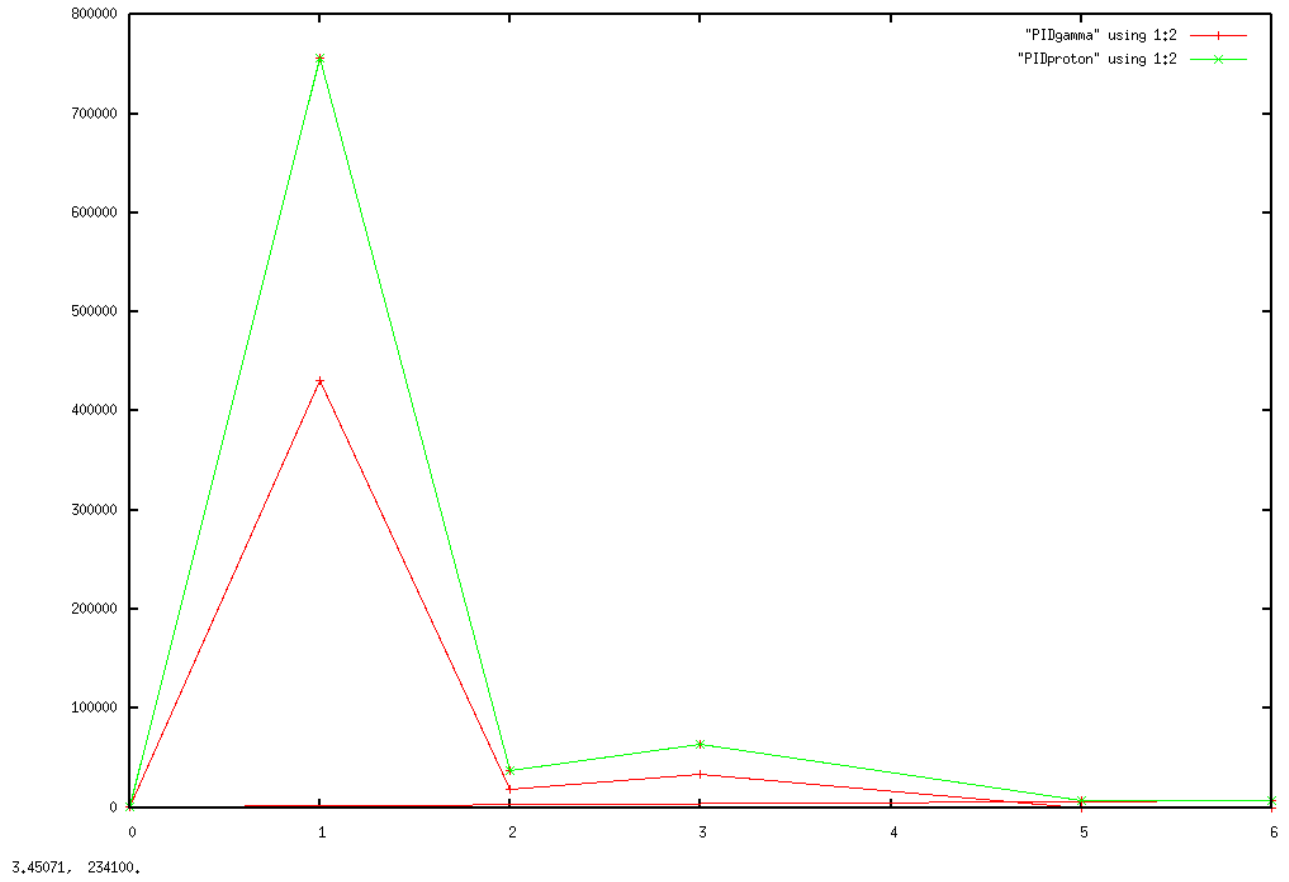
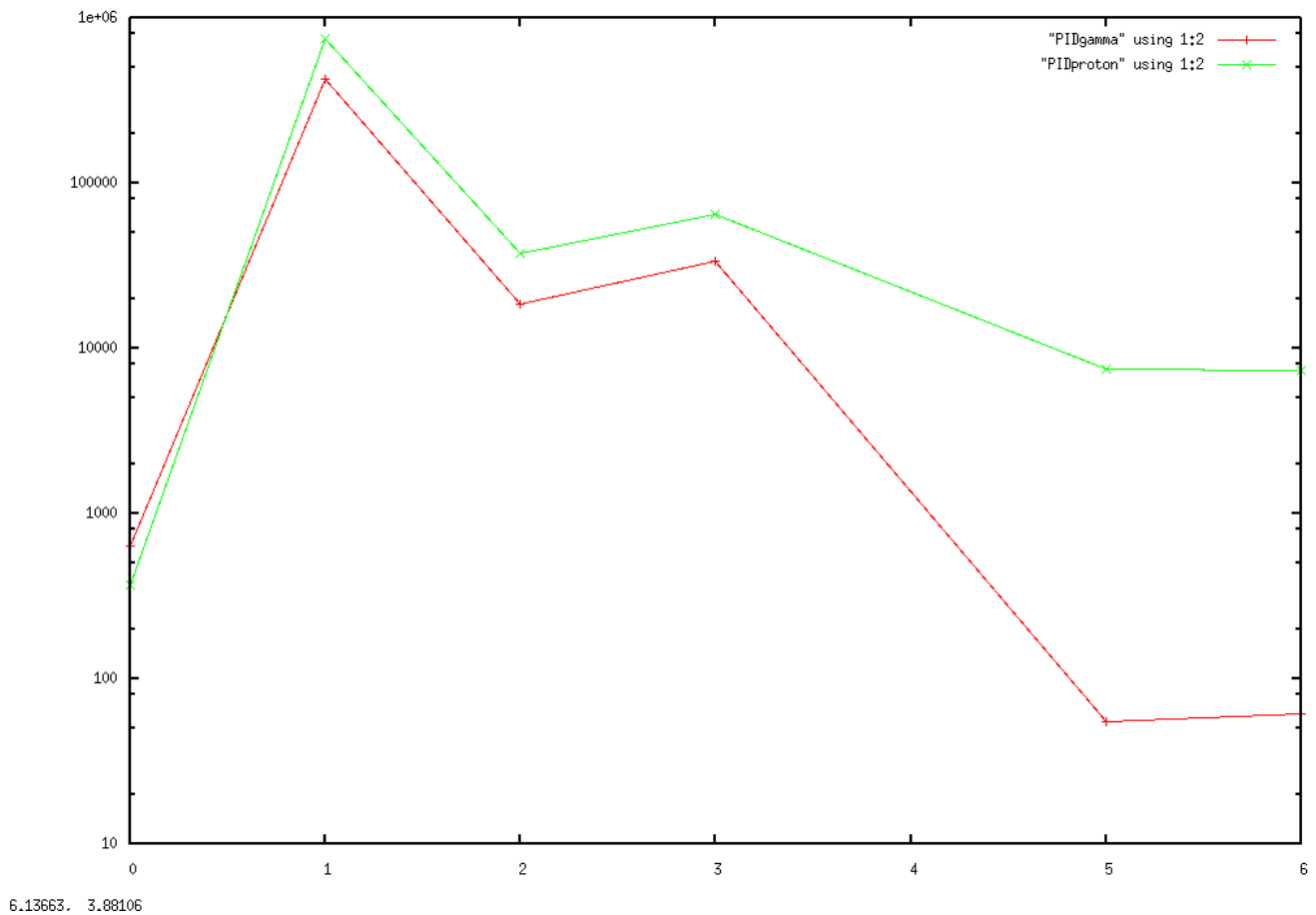


Image plotted with Log Scale

In the following plot the same data set used above is plotted with a log scale function in order to make more subtle differences stand out. As a result we find that daughter particle types produced by both gamma and proton parents is similar, although through use of the log scale function the differences between particle types five and six, the muon particle and antiparticle, μ^\pm , is made more clear.



7.5 Particle Distance from Center

In the following images we find a representation of the individual particle distances from the

center plotted as a function of number of particles versus particle distance. On the image portrayed directly below we find that the distances of particles, i.e muons, produced by different parent particles is very close regardless of the fact that they are produced by different parents. As a result we can label particle distance from the center as a fairly inaccurate means of distinguishing the parent of an interaction, but a valuable means of differentiating between daughter particle types produced.

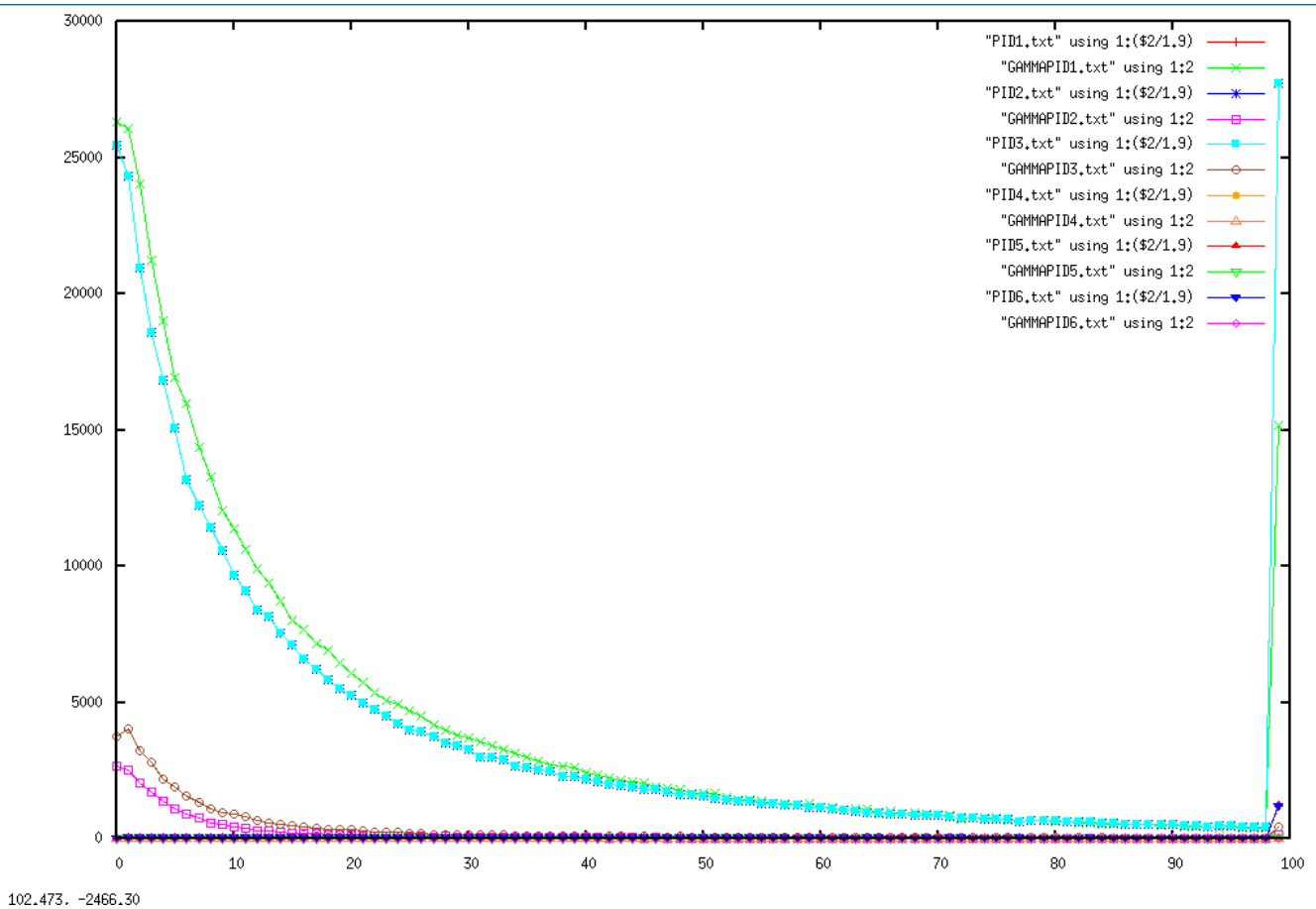
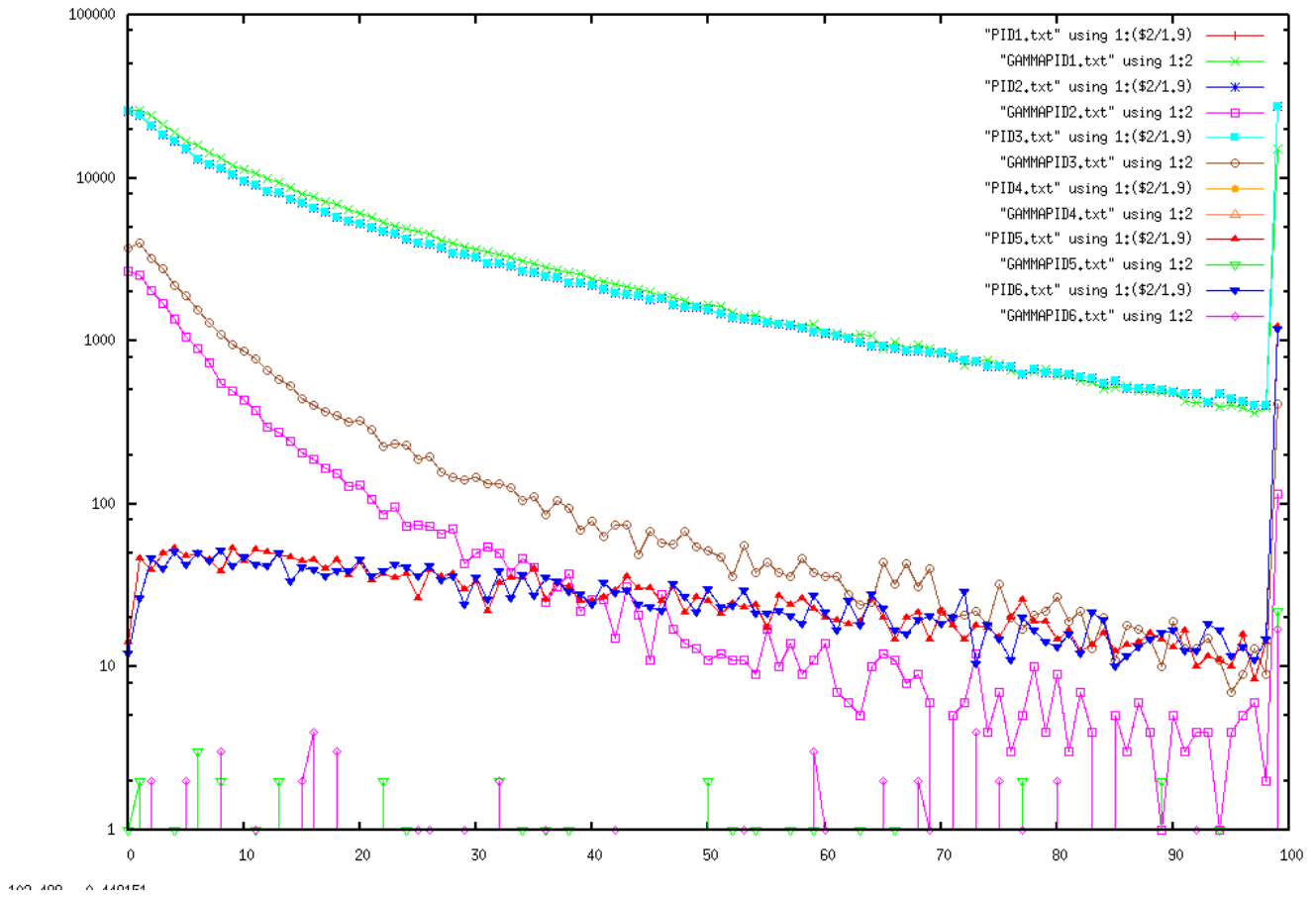


Image Plotted with Log scale Function

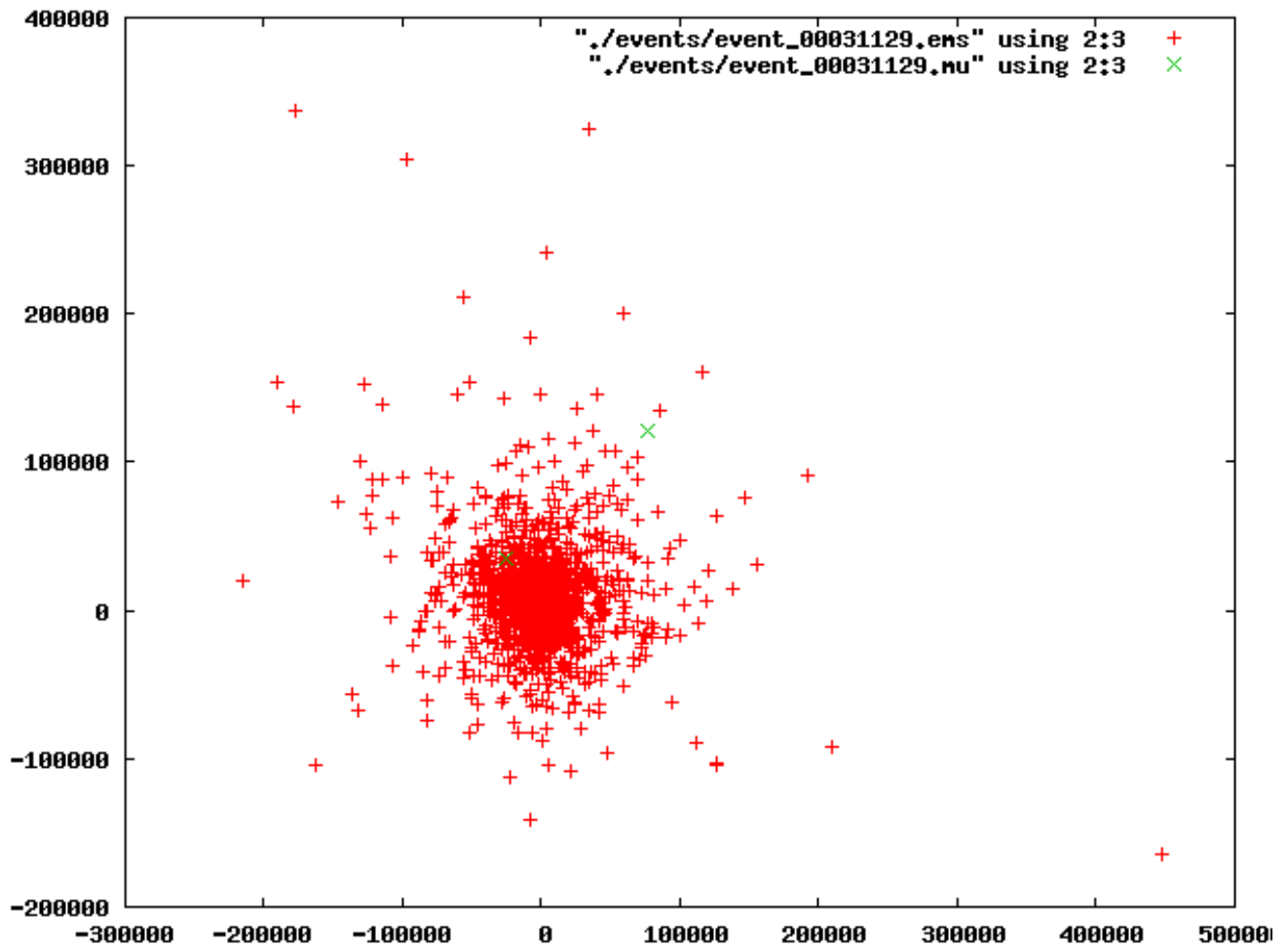
The plot below represents the same data table as above although with an added log scale function used to make subtle differences between our plotted data easier to interpret. Likewise the

linear regression of each particle type versus distance from center is made more clear.



7.6 Proton Shower Images

Following are a series of proton plots produced in CORSIKA and plotted in GNUplot, each represents an individual shower with differing energies. Through evaluating these plots we can begin to see differences between different interactions, i.e. muon production in proton showers⁷, in the case of the following images.⁸

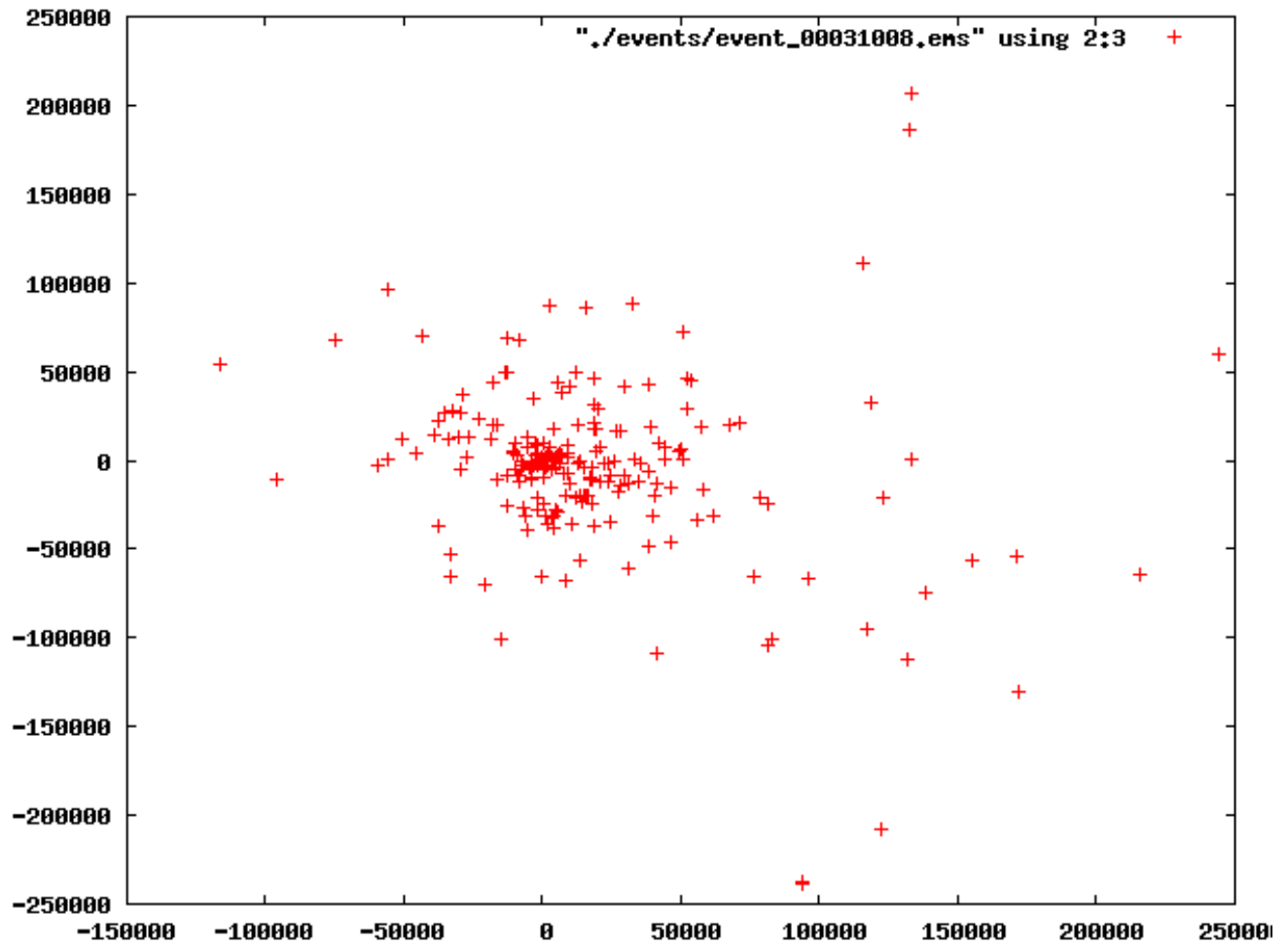


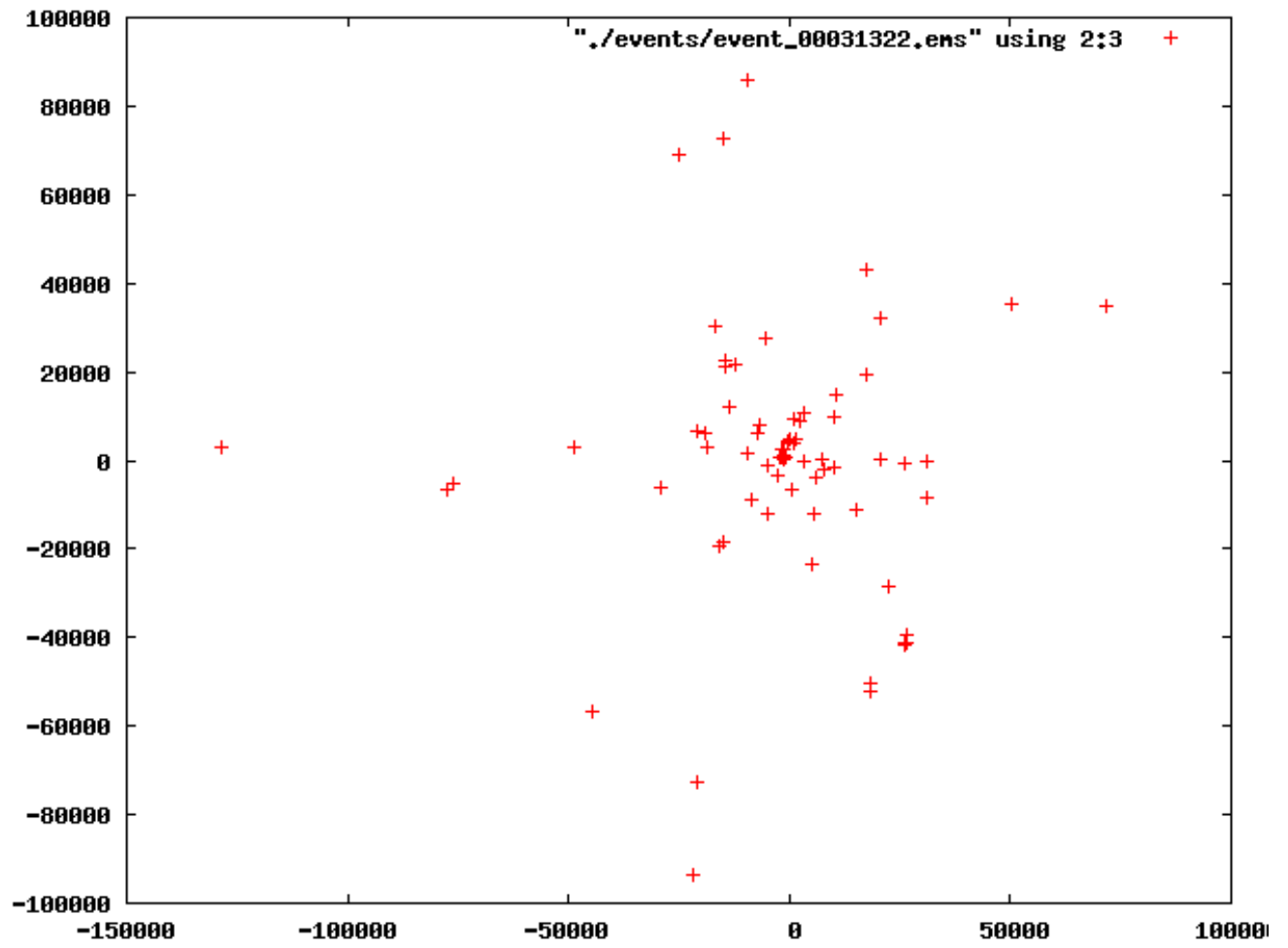
⁷ Muon particles represented in green in plots.

⁸ All GNUplot images original product of data produced by CORSIKA v. 6900

7.7 Gamma Shower Images

The following images contain data relative to gamma particle interactions; once again we find some visual differences between gamma and proton induced interactions. Note in the following images the lack of muons and overall difference in number of daughter particles.





8. *Future Work*

This project is still in progress. In the future we would like to:

- Input data and actually determine whether the interaction is a proton or a gamma. Right now we only have characteristics; we do not actually have solid conclusions that the interaction we are looking at is a proton or a gamma
- Optimize our coding to produce images in our GNUplots that will be easier and more efficient to analyze
- If possible we would like to feed our coding with real world data and compare both the real world data and data from the CORSIKA source code to determine if results are the same in both situations.

9. *Conclusion*

With our research we have discovered that our coding and the CORSIKA source code will help us develop plots and graphs that will distinguish between characteristics of both Hadronic and Electromagnetic air showers. Some characteristics we have discovered in our plots and graphs are energy amounts, the spread and compression of the energy amount, distance away from the center, and daughter particles. With the characteristics that we have gathered from our research we will be able to determine the differences between Hadronic and Electromagnetic air showers. We have found that the most significant original achievement made as a result of our project is relative to determining specific characteristics of different air shower events.

10. Acknowledgments

In consideration of support received from external sources and people we would like to accredit the success of this project to:

- John Pretz
- Brenda Dingus
- Michelle Thomsen
- Philip Sanchez

11. Bibliography

10.1 Internet Resources

Carlson, Shawn, "Counting Particles from space". Scientific American February 2001:

1-2

Wales, Jimmy. "Statistics". Wikimedia Foundation, Inc.. April 2010 <http://en.wikipedia.org/wiki/Statistics#Statistical_methods>.

Pierog, Tanguy. "CORSIKA an Air Shower Simulation Program". Karlsruhe Institute of Technology . April 2010 <<http://www-ik.fzk.de/CORSIKA/>>.

Wales, Jimmy. "Bethe Formula". Wikimedia Foundation, Inc.. April 2010 <http://en.wikipedia.org/wiki/Bethe_formula>.

Wales, Jimmy. "Lorentz Factor". Wikimedia Foundation, Inc.. April 2010 <http://en.wikipedia.org/wiki/Lorentz_factor>.

Wales, Jimmy. "Muon". Wikimedia Foundation, Inc.. April 2010 <<http://en.wikipedia.org/wiki/Muons>>.

Wales, Jimmy. "Particle Physics". Wikimedia Foundation, Inc.. April 2010 <http://en.wikipedia.org/wiki/Particle_physics#Subatomic_particles>.

Wales, Jimmy. "List of Particles". Wikimedia Foundation, Inc.. April 2010 <http://en.wikipedia.org/wiki/List_of_subatomic_particles>.

