Creating a Research Program - Part I

Computer simulation models are presently used in all aspects of scientific research. They are now considered to be, along with theory and experiment, one of the three main branches of scientific inquiry (Perricone, 2006). One area that computer simulation has especially been valuable for is the modeling of black holes. This paper will discuss different black hole simulation applications and will preview the initial development of my own Visual C++ black hole simulation program.

One of the earliest physics C++ modeling programs was Geant4, developed in 1994 for use in the European Organization for Nuclear Research (CERN) Hadron Collider in Switzerland (Perricone, 2006). The name Geant4 is a contraction of the words “geometry” and “tracking” in reference to the paths of the particles and the ability to track them. The program was written in C++ because of its object oriented nature, and that CERN is attempting to detect the most elusive objects in the universe, subatomic particles. The main responsibility of the Geant4 program is run simulations on the paths of particles in order to determine the most optimal usage of the equipment. Geant4 is instrumental in saving costs for research. If the physicists at CERN can first run a simulation of the path of a particle trajectory before they actually turn on the Hadron Collider, they can save valuable time and expenses. The simulations can help them determine the best location to place the detectors and related equipment.

A more recent simulation program, known as Cactus Code, is a three dimensional C++ open source platform developed by Louisiana State University. The program took over 10 years to write and was financed in part by the National Science Foundation and NASA (Schnetter, Tao, 2006). Current applications include not only Numerical Relativity black hole simulations, but also coastal modeling and petroleum engineering. The structure is based on a central application known as “flesh” that controls a series of plug in sub-applications known as “thorns”. The flesh is autonomous and acts a service library, but the thorns need the flesh to perform operations. A thorn can be changed or modified to suite the needs of the programmer. Some examples of thorns for three dimensional General Relativity black hole simulations are boundary conditions, state equations (locations), gravity wave models, gamma ray bursts, exotic stars, and error controlling (Schnetter, Tao, 2006). Some of the Cactus Code design parameters include parallel processing capabilities, flexible build systems, and multiple grid functions. The program allows for resolution of images from 10,000 km to 100 m in 100 seconds with 16 levels of refinement (Schnetter, Tao, 2006). However, in order for Cactus Code to perform the complex three and even four (including time) dimensional calculations needed for General Relativity, there needs to be a large amount of computing power. For example, the grid supporting it will need to perform a Peta (quadrillion) floating-point operations per second (PFLOP) for a 10 minute run time, and will need a tera(trillion) bytes (TB) of memory available, with at least 650 mega(million) byte of memory per core computer (Schnetter, Tao, 2006).

Related to C++ black hole modeling programs is a simulation for the time intervals of the Einstein-Dirac Equation (Hiptmair, Zeller, 2009). This equation is used to calculate the interaction of a sub-atomic particle with the gravitational field (Friedrich, Kim, 2008).

It combines the equation of General Relativity, which describes the very large, with the Dirac Wave Equation, which describes the very small. In geometrical form, the General Relativity and Dirac Wave Equations are respectively:

$Ric_g - 1/2 S_g g = \varepsilon/4\, T(g, {}_\psi)$, $D_g\psi = \lambda\psi$

Where:

Ric_g = the Ricci Tensor, which describes the curvature of space time due to gravity

$S_g g$ = the scalar or non-directional single point curvature, similar to a “bend” or “crease”

ε = Relative Permittivity, or effect of an electric field = $8.854\ 187\ 817\ldots \times 10^{-12}$ F/m (charge/meter)

D_g = the Dirac Operator, describing the movement of a particle in a gravitational field

ψ = the particle energy field

λ = the eigenvalue or change in magnitude value of the field vectors (directions)

$T(g, \psi)$ = the energy momentum field tensor. This describes the energy momentum field coordinates and is determined by the equation:

$$T(g, \psi)((X, Y) := (X \cdot \nabla gY \psi + Y \cdot \nabla gX\psi,\psi)$$

Which describes the energy and momentum of the field in X and Y directions.

(Friedrich, Kim, 2008).

The Einstein-Dirac Equation predicts that in a black hole the electric charge is so strong that it balances the gravitational force and therefore prevents subatomic particles from being pulled into its edge or event horizon (Friedrich, Kim, 2008). The Einstein-Dirac Equation C++ application program runs on parallel computing systems and its user interface can run on shell scripted files. Data obtained from the application is saved as a binary text file. Once saved, the data can be exported to a simulation program.

Black holes may not just be the domain of the very large, but may actually exist on a microscopic level (Cavaglia, Cremaldi, Godang, Jenkins, Summers, 2008). Theoretically, micro black holes could be created in the presence of extremely high energy particle collisions, such as the 14 trillion electron volt (TeV) mass proton-proton collisions at the CERN Large Hadron Collider (LHC) (Cavaglia, et. al., 2008). A micro black hole may occur at these high energies if the magnitude of the energy causes the escape velocity of any particle to be greater than the speed of light, c = 300,000,000 meters per second. This energy level is calculated to be normally in the tera TeV range of the CERN LHC. The University of Mississippi has developed a FORTRAN and C++ program known as the Collider Gravitational Field Simulator for black holes (CATFISH) used to generate simulated LHC events that may lead to the creation of the micro black holes. The program was originally written in FORTRAN but has shifted to C++ due to the desire for greater simulation capabilities as opposed to collecting data.

My Visual C++.NET project will be to create my own black hole simulation program. This program will be based on research I have conducted on existing black hole simulations. The following is a series of screen shots showing the progression of the project. The first is the initial set up of the program. I have created the dynamic link libraries (DLL) for the program that will access the programming archives. These dll files are also shown below due to the fact that they may be a little difficult to read on the screen shot.

#using<mscorlib.dll> accesses the core library

#using<System.dll> accesses the system library

#using<System.Window.Forms.dll> accesses the system Windows forms

(Neiman, p. 11).

```
// Black Hole.h

#pragma once

using namespace System;

namespace BlackHole {

	public ref class Class1
	{
		// TODO: Add your methods for this class here.
	};
}

#using <mscorblib.dll>
#using <System.dll>
#using <System.Windows.Forms.dll>
using namespace System;
using namespace System::Windows::Forms;
int _stdcall WinMain()
{
	MessageBox::Show(S"Black Hole Program");
	return 0;
}
```

This second screen shot shows the progression in the program. The command *String* is used to create a string value for the area of the black hole. This may be modified to include the actual calculation of the disk area the second command is to create byte object. This byte object is used to set minimum and maximum values for bytes and then to convert these values to strings

(Neiman, p. 65). The third command is a string builder to give any decimal value entered into the program to have a two place precision (Neiman, p. 81). These commands are initial types and may continue to be modified and updated as the program design progresses. The idea behind creating them was to have a general framework to base the content of the rest of the program on.

```
String *str;
str = S"Area of black hole horizon";
MessageBox::Show(area.ToString(), str);
//this is for the user to enter the area of the black hole.//
//I will also include a radius calculation later in the program//
System::Byte btObject1;
byte btObject2;
btObject1 = 0x0F;
btObject2 = 0xF0;
MessageBox::Show(btObject1.MinValue.ToString(), S"Minimum Byte value");
MessageBox::Show(btObject1.MaxValue.ToString(), S"Maximum Byte value");
//used to set s minimum and maximum byte value and convert to strings//
StringBuilder * sbDecimal = new StringBuilder();
String * strBuffer = S"{0:f2}";
sbDecimal->AppendFormat(strBuffer,_box(decAnswer));
MessageBox::Show(
      sbDecimal->ToString(),
      S"2-digit Decimal only"

      );
//gives decimals 2 digit precision//
```

Conclusion

Black hole simulations are an effective method for scientists to study their properties, as long as the necessary equations for General Relativity and Quantum Mechanics are included within the framework of the program. This is the initial phase of my Visual C++ program; one that will experience numerous modifications and revisions. A substantial portion of these revisions will be related to the formulas and mathematical calculations needed to assure that accurate data may be collected from the simulation model. In addition, due to the fact that this program will be created and compiled on a personal computer and not a main frame or a shared parallel network, it will contain a limited amount of programming code and graphical animation. However, the program will still need to fulfill the proper requirements for a simulation model. This is the substantial challenge that I have set before me.

References

Cavaglia, M., Cremaldi, L., Godang, R., Jenkins, C.M., Summers, D. (August 18, 2008). *Adaptation of a Fortran-based Monte-Carlo microscopic black hole simulation program to C++ based root.* Abstract submitted for the SES08 meeting of The American Physical Society.University of South Alabama, University of Mississippi. Retrieved August 18, 2009 from http://absimage.aps.org/image /image/MWS_SES08-2008-000195.pdf

Friedrich, T. Kim, E.C. (February 1, 2008). *The Einstein-Dirac Equation on Riemannian Spin Manifolds.* Humboldt-Universit¨at zu Berlin, Institut f¨ur Reine Mathematik, Ziegelstraße 13a, D-10099 Berlin, Germany. Retrieved August 18, 2009 from http://http://arxiv.org/PS_cache/math/pdf/9905/9905095v1.pdf

Hiptmair, R., Zeller, B. (January 26, 2009). *Numerical simulation of the Einstein-Dirac equations in a spherically symmetric spacetime.* Retrieved August 18, 2009 from http://www.sam.math.ethz.ch/~hiptmair/Projects/DiracEinstein.pdf

Neiman, A. (2006). *Hit the ground running with visual C++.NET.* Clark, Ed. Thomson Delmar Learning. ISBN: 1401878806

Perricone, M. (November 2006). GEANT4: the physics simulation toolkit. *Symmetry Magazine.* 2(9). Retrieved August 18, 2009 http://www.symmetrymagazine.org/pdfs/200511/geant4.pdf

Schnetter, E., Tao, J. (2006). *Cactus framework: scaling and lessons learnt.* Center for Computation & Technology, Departments of Computer Science & Physics, Louisiana State University. Retrieved August 18, 2009 from http://www.cct.lsu.edu/xirel/Documents/BlueWaters_Allen.pdf

Creating a Research Program - Part II

This paper is a continuation of Part I. It gives additional information on black holes and the simulation models sued to study them, along with the further developments of my Visual C++ black hole simulation program taken from the Neiman text, chapters four to seven. This includes arrays and pointers, managed and value classes, characters and strings, and classes and data abstractions.

Numerical Relativity is one of most common methods to study the properties of black holes, including black hole collisions and binary or twin black holes. It is used primarily to study the movements of four dimensional space-time. For example, the “cartoon method” simplifies these calculations by leaving out non-symmetrical dimensional changes. In this method, there are no changes in dimensions, and the objects used are kept as simple as possible, with no need or requirements to be complex. The model is symmetrical in four dimensions and can be “flipped” in a mirror image, shown symbolically as $U(1) \times U(1)$ $x = y, z = w$ (Yoshino, Shibata, 2009).

The *x* (horizontal) is equal to the *y* (vertical) and the *z* (space) is equal to *w* (time). The simulation program CATFISH can be used in Monte Carlo (MC) code to calculate 14 trillion electron volt (TeV) proton to proton collisions in the Large Hadron Collider at CERN. CATFISH takes into consideration such things as the Hawking Evaporation of black holes, which aids in creating accurate information on the creation, evolution and decay of black holes (Bock, Humanic, 2008). The CATFISH process is the black hole mass is calculated from a cross section, then the black hole is decayed through the equations of Hawking Evaporation, and finally subatomic particles that are emitted are measured.

There are different models that can be programmed into CATFISH. These models include a No Gravitational Loss model (NGL) that is used to calculate micro black holes in four dimensions, with three space and one time. The equation with mass M and circumference C for this model is:

H_D ç $C/2\pi rh(M)$ ¹ 1

With $rh(M)$ = the Schwarzschild Radius, which is the minimum radius needed for a black hole.

$rh(M) = (16\pi G_D M /(D - 2)\Omega_{D-2})^{1/(D-3)}$

Ω_{D-2} = the volume of the black hole sphere

G_D = the gravitational constant = $6.6742x10^{11}m^2/kg^2$

This has to be smaller than the Schwarzschild Radius for a black hole to form.

The black hole black disk cross section equation is:

$\sigma_{BD} = \pi R^2(s, n)$È$[R(s, n) - b]$

Where R = the horizon radius and depends on extra dimensions n and center of mass energy $\sqrt{s}$

b = the impact parameter of two colliding particles.

for this model to work, the black hole mass = central particle mass energy (Bock, Humanic, 2008).

The Yoshino-Nambu model (YN) measures the event horizon of the black hole at the collision of two shock waves, which is known as the Trapped Surface approach. For this model, the equation that defines the conditions for the formation of a black hole is:

H_D **ç** $[V_{D-3}/ \Omega_{D-3} r_h{}^{D-3}(s, n)]^{1/(D-3)}$ $^{\mathbf{1}}$ 1

Where Ω_{D-3} = the volume of the three dimensional sphere

V_{D-3}= the characteristic three dimensional volume of the system

A cross section of the black hole is given by:

$\sigma_{BHproduction} = F(D) r^{2}{}_{h}\ (s, n)$

Where $F(D)$ = numerical factor close to unity.

In this case, the black hole mass < central particle mass energy because the black hole is emitting gravity particles (gravitons) (Bock, Humanic, 2008).

The third and final simulation model is the Yoshino-Rychkov model (YR), which is an upgraded YN model. The improvements have to do with the fact that the edge or horizon of the black hole is created by cutting a slice of the conical shock wave path that light makes in the time dimension, known as the future light cone. This gives a smaller black hole mass than the previous models (Bock, Humanic, 2008).

Hawking used a novel approach for calculating the gravitational fields from a black hole. He assumed that the black hole was closed and that information could not lost within the singularity, which is the point of greatest gravity within the center (Hawking, 2005). He used what is known as a Euclidian path Integral equation to measure the gravitational effect. This integral is used to measure the path of a particle over a multidimensional surface. The equation for the path at infinity is shown below:

$Z(\beta) =$ **ç** $DgD\varphi e^{-I[g,\varphi]} = Tr(e{-}^{\beta H})$

With the wave function $\Psi[h_{ij}, \varphi, t]$

Where:

β= the length of the path

Dg= the change in gravitational field

$Tr(e^{-\beta H})$ = the trace of the path

$h_{ij,}$ = three dimensional coordinates

φ= the matter field

T and t= time at infinity

This equation shows the Schwarzschild Radius exists in a spherical (anti-de Sitter) space, and the black hole is eternal if large. If the space is small, flat the black hole will be microscopic and evaporate (Hawking, 2005). Another conjecture put forth by Visser is that a black hole does not need an edge, or event horizon to evaporate. If true, this means a microscopic black hole can evaporate and give off energy (Visser, 2008).

I continue to develop my Visual C++ black hole simulation program. First, I will create a two dimensional variable array to use for the dimensions of the black hole. The code here is to set up the array:

```
int_stdcall WinMain ()
{
ArrayList * al2Dint = new ArrayList();
ArrayList * rows[] = new ArrayList * [3];
```

(Neiman, p. 159).

This code is to set up the variable objects.

```
for (int i = 0; i < rows->Length; i++)
{
        rows[i] = new ArrayList();
        for (int  j = 0; j < 2 + i; j++)
```

```
            rows[i]->Add(_box(100 * i + 10 * j));

      al2Int->Add(rows[i]);

}
```

(Neiman, p. 160). Here is the screen shot of my code used to create the array, taken from Neiman:

```
#using <mscorlib.dll>
#using <System.dll>
#using <System.Windows.Forms.dll>
using namespace System;
using namespace System::Collections;
using namespace System::Windows::Forms;

int __stdcall WinMain()
{

      // Here is the primary ArrayList (the 2-dimensional array):
      ArrayList * al2DInt = new ArrayList();

      ArrayList * rows[] = new ArrayList * [3];


      for (int i=0; i<rows->Length; i++)
      {

            rows[i] = new ArrayList();

            for (int j=0; j<2+i; j++)
                  rows[i]->Add(__box(100*i + 10*j));

            al2DInt->Add(rows[i]);
      }

      IEnumerator * ienRows = al2DInt->GetEnumerator();

      String * str = S"";

      while (ienRows->MoveNext())
      {

            IEnumerator * ienColumns =
                  (__try_cast<ArrayList *>
                        (ienRows->Current)
                  )->GetEnumerator();

            while (ienColumns->MoveNext())
                  str = String::Concat(
                        str,
                        ienColumns->Current->ToString(),
                        S" "
                  );
            str = String::Concat(str, S"\n");
      }
      MessageBox::Show(str, S"2D Managed Array Variable Size");
```

```
	return 0;
}
```

Here is a second screen shot showing how the array will be accessed, again from Neiman (Neiman, p. 162).

```
#using <mscorlib.dll>
#using <System.dll>
#using <System.Windows.Forms.dll>
using namespace System;
using namespace System::Collections;
using namespace System::Windows::Forms;

ArrayList * make2DArrayList(int rows, int cols)
{

	ArrayList * al2DInt = new ArrayList();

	ArrayList * dimension1[] = new ArrayList * [rows];

	for (int i=0; i<dimension1->Length; i++)
	{

		dimension1[i] = new ArrayList();

		for (int j=0; j<cols; j++)
			dimension1[i]->Add(__box(100*i + 10*j));

		al2DInt->Add(dimension1[i]);
	}

	return al2DInt;
}

void show2DArrayList(ArrayList * al2DInt, String * name)
{

	IEnumerator * ienRows = al2DInt->GetEnumerator();

	String * str = S"";
	// The MoveNext method obtains the current row:
	while (ienRows->MoveNext())
	{
		IEnumerator * ienColumns =
			(__try_cast<ArrayList *>
				(ienRows->Current)
			)->GetEnumerator();

		while (ienColumns->MoveNext())
			str = String::Concat(
				str,
				ienColumns->Current->ToString(),
				S" "
			);
		str = String::Concat(str, S"\n");
	}
	MessageBox::Show(str, name);
}

int __stdcall WinMain()
{
	ArrayList * al2DInt = make2DArrayList(3, 4);
	show2DArrayList(al2DInt, S"Original 2D ArrayList");

	Int32 val = Int32::Parse(
		(
			(__try_cast<ArrayList *>(
				al2DInt->get_Item(1)
				)
			)->get_Item(2)
		)->ToString()
	);
	MessageBox::Show(val.ToString(), S"Array Value: Row 1 Column 2");

	(__try_cast<ArrayList *>(
		al2DInt->get_Item(1)
		)
	)->set_Item(2, __box(5000));
	show2DArrayList(al2DInt, S"After changing Row 1 Column 2 ==> 5000");

	return 0;
}
```

The array will also need to be accessed with pointers. I also need to create an object class called "Black Hole". This object will also contain polymorphism, for functions to be used in different ways, and inheritance, to create new classes. The strings for the program will also need to be concatenated and parsed, which means to connect and proofread two strings. A data abstraction defines the classes by how they are similar. Here is the screen shot of the class "Black Hole". I have not added the polymorphism classes yet, but wanted to show the class and pointers (Neiman, p. 160).

```
#include "Blackhole.h"

int __stdcall WinMain()
{
	// Create an object of Black Hole class,
	// Create a pointer and use the new Keyword.
	// The name of the object is 'B'.

	Black Hole * b = new Black Hole();

	MessageBox::Show(b->ToString(), S"Black Hole");

	return 0;
}
```

Conclusion

This continuation of my black hole simulation project includes setting up an array, accessing the array, and creating a class for the black hole. The next main step will be how to include the calculations for General Relativity in the program, and how to continue developing the objects and classes needed for the project.

References

Bock, N. ,Humanic, T.J. (November 15, 2008). *Quantitative calculations for black hole production at the Large Hadron Collider*.(v2)Journal reference: Int.J.Mod.Phys.A24:1105-1118,2009. Retrieved August 27, 2009 from http://arxiv.org/PS_cache/arxiv/pdf/0806/0806.2111v2.pdf

Hawking, S. (2005). Information loss in black holes. *Physical Review D* **72**: 084013. Retrieved August 27, 2009 from http://arxiv.org/PS_cache/hep-th/pdf/0507/0507171v2.pdf

Neiman, A. (2006). *Hit the ground running with visual C++.NET.* Clark, Ed. Thomson Delmar Learning. ISBN: 1401878806

Visser, M. (August, 2008). *Black holes in general relativity.* School of Mathematics, Statistics, and Operations Research, Victoria University of Wellington. Retrieved August 27, 2009 from http://arxiv.org/PS_cache/arxiv/pdf/0901/0901.4365v3.pdf

Yoshino, H. Shibata, M. (July 16, 2009). *Higher-dimensional numerical relativity: formulation and code tests.* Department of Physics, University of Alberta, Yukawa Institute for Theoretical Physics, Kyoto University, Kyoto, 606-8502, Japan. Retrieved August 28, 2009 from http://arxiv.org/PS_cache/arxiv/pdf/0907/0907.2760v1.pdf

Analysis of .NET classes

A class is defined as an abstract data type (ADT) that contains, controls, and maintains the information contained in a object oriented field (Neiman, p. 187). An ADT combines the operations with the data that it includes (Lajoie, Lippman, Moo, 2005). A class inherits certain attributes that represent it, and they can combine information from other objects, known as aggregation, if necessary. To create a managed class, there must be a public declaration in the program. For example, the class is stored in an *.h* file, and the declaration has a *_gc* prefix. The coding to create a managed class called *Black Hole* would look something like this:

```
#pragma once// this is necessary for a .h file//
#using<mscorlib.dll>//accesses the core library//
#using<System.dll> //accesses the system library//
using namspace System:
using namspace System:Windows:Forms
pblic _gc class Black Hole
```

(Neiman, p. 190).

The public declaration sets up the class to be visible, managed, and to collect garbage files (*gc*). However, there will be some data that will not be public, so there needs to be a private declaration after the brackets under public. Once the class has been declared as both public and private, there needs to be a series of objects used to help define the class. For example, in the case of Black Hole, the objects may be:

```
this-> gravitational pull      = S"Defines";
this-> event horizon           = S"Edge";
```

where this-> is "the address of the object" (Neiman, p. 192).

The class will need to be converted to a string, with each defining object concatenated to the each other.

```
String * Black Hole::ToString()
```

```
{
Return String::Concat(
        S"The gravitational pull is ",
        this-> gravitational pull
```

and so one for each object of the class. After the class has been created and defined, it needs to be declared. The #include *"BlackHole.h*" needs to be added at the top of the program above

```
int_stdcall WinMain()
{
```

There are a variety of different operations and functions that can be created and utilized when it comes to classes and their objects. For example, a class can reference a cost function that declares the price (Neiman, p. 201). The class use a *_property* notation or custom class names and is set up in the following manner:

```
 _property void
get_Radius(Single Value);
_property Single
get_Radius( ) (Neiman, p. 204).
```

The positive attribute about an array is that is can convert the data to a series of pointers (Mayne, p. 38). It is beneficial for handling large amounts of data. The arrays can group the data, and the pointers can be set to find the location. There is no need to declare variables at the top of the program. The pointer name includes the name of the variable with a * following it (Mayne, p. 41). The class can also have + overload operators using *Record* to increment values, which will need to be declared publically *Record.h* (Neiman, p. 213). In addition, a class can inherit other specific codes that are separated from the more general program (Neiman, p. 222). Lastly, the class can have what is known as an abstract base, which can be used for a variety of formats (Neiman, p. 239).

Conclusion

In conclusion, Microsoft Visual C++.NET classes are crucial elements that help to define program structure and give it a framework for the creation of specific objects. When it comes to handling large amounts of data, arrays and pointers are the most efficient method. Managed classes give the programmer an opportunity to create global variable that can be used throughout the program. An abstract data type (ADT) is also beneficial because it allows consolidation and simplification of the data and information contained in a class and object. Finally, a string configuration can convert the data of an object tot a text file. Depending on the specific application, all of these .NET classes can have an important place in a developer's tool kit.

References

Lajoie, J. Lippman, S.B., Moo, B (2005). *C++ primer*. Library of Congress Cataloging-in Publication Data. ISBN 0-201-72148-1. Retrieved September 11, 2009 from http://my.safaribooksonline.com/0201721481/part03

Mayne, R. (August, 2005). *Introduction to windows and graphics programming with visual C++.NET*. University at Buffalo, State University of New York, USA. ISBN: 978-981-256-455 9. Retrieved September 11, 2009 from http://www.worldscibooks.com/compsci/5776.html

Neiman, A. (2006). *Hit the ground running with visual C++.NET*. Clark, Ed. Thomson Delmar Learning. ISBN: 1401878806

Creating a Research Program - Part III

This paper is a continuation of my Visual C++.NET black hole simulation program. This portion of the research will include concepts from chapters eight and nine of the Neiman text, specifically Managed Arrays, .NET Windows Form Controls, and Advanced .NET Form Components.

Black hole simulations are considered to be part of a rapidly growing branch of Astronomy known as “Virtual Astronomy”. This new type of Astronomy is a necessity when there is no practical method to perform a real life test, such as in the case of a black hole. The research simulation conducted by Bakala, Ccaron;ermák, Hledík, Stuchlík, and Truparová Plšková, (2005) consists of a model that predicts how the universe will look to an observer next to a black hole. In other words, it predicts how light is being bent around the edge or event horizon by the extreme gravitational fields. The type of simulation will need to be performed in what is known as Schwarzschild-de Sitter spacetime, which is a four dimensional black hole spherical gravitational field in a symmetrical universe with a positive (repulsive) Cosmological Constant (Bakala, et. al., 2005). The Cosmological Constant, denoted as Λ, is a term that Einstein developed to balance out his General Relativity Equation so that the universe could be static and non-expanding. Since then it has found some practical uses in General Relativity Theory. When it is positive as in the above example, the universe is expanding. The light is bent in an extreme fashion known as gravitational lensing, where the force around the event horizon acts as refractive lens to an observer (Bakala, et. al., 2005). This is a common method for astronomers to detect black holes in space.

Black hole simulation models are also becoming popular in the general media. The Denver Museum of Nature & Science, in collaboration with NASA, presented a 20 minute virtual tour of the inside of a black hole titled *Black Holes: The Other Side of Infinity* (Wanjek, 2006). The model used in the tour was created by Andrew Hamilton of the University of Colorado at Boulder with data obtained from the NASA Swift satellite, which was designed to detect gamma ray bursts (Wanjek, 2006). A gamma ray burst is the evidence for creation of a new black hole. This is first actual simulation of what may be going on inside of a black hole, where luminous gases are formed and concentrated into a spiral shape under the force of extreme gravity. It has been theorized that most galaxies have a black hole at their center (Wanjek, 2006).

Black holes may also exist as pairs, similar to binary or twin stars that feed off each other's gravitational forces. When two black holes get close enough to collide, their event horizons can crash into one another and can form what are known as gravitational waves (Hannam, 2009). These gravitational waves are very faint, and require a very sensitive series of instruments to detect them. One such experiment is the Laser Interferometer Gravitational Wave Observatory conducted by the Massachusetts Institute of Technology (MIT) and the California Institute of Technology (CalTech). The observatory consists of a group of lasers contained in an underground tunnel. The laser light signals are bounced off a mirror at a 90 degree angle to a detector. The detector can determine movements in the detector plates of millionths of an inch. This is why the experiment needs to be performed underground and away from any seismic activity. In 2009, the LIGO detectors were enhanced to receive signals in the 30 Hertz (cycles per second) range, which was determined by simulation models to be the optimal frequency for gravitational waves. The equation used to calculate the gravitational wave signal is:

$$h_{\ell m} \equiv \ddot{e}_{-2}Y_{\ell m},\ h\eth = \text{ç}^{2n}_{0}\ d\varphi\, \text{ç}^{n}_{0}\ (h_{-2}Y_{\ell m})\ \sin\theta d\theta$$

Where:

$n = \pi$

$h_{\ell m}$ =the gravitational wave harmonics

$\text{ç}d\varphi$ = the area under the gravitational three dimensional sphere

$\text{ç}\ (h_{-2}Y_{\ell m})\ \sin\theta d\theta$ = the area and period of the wave harmonics (Hannam, 2009). Current computer simulations of the LIGO detectors are limited by the accuracy of their predictions.

The paths of the waves may not be in agreement with experiment and can create discrepancies in the models.

Black holes have applications in many different areas of Physics, Astronomy, and Applied Mathematics, along with Computational Science and Modeling and Simulation. For example, the Thermodynamics or energy of black holes is Stephen Hawkings' primary research area.

Hawking has calculated that black holes evaporate at a given rate, determined by the surface temperature:

$T \equiv \partial M / \partial S]_Q = r_+ - r_- / 4\pi\, r_+^2 = \kappa / 2\pi$

Where:

$\partial M / \partial S]_Q$ = the change in mass divided by the change in energy of a black hole

$r_+ - r_- / 4\pi\, r_+^2$ =the change in black hole radius divided by the energy

κ = the surface gravity or gravitational force of a black hole (Kiritsis, 2007).

An effective use for a black hole simulation model would be to run a computational simulation to verify the evaporation rate of a black hole, given its size and surface gravity.

A managed array uses [ ] brackets to create objects (Neiman, p. 101). These objects can be manipulated and modified to suite the needs of the program. For example, I wanted to create a managed array for my black hole simulation program, but I needed to change the variables of the array, which I had shown in Activity Three. At this stage of the project, I will construct a graphical user interface (GUI) for the users to interact with the program. I needed to declare a class called *Button* for the program to interact with. The following is the declaration used by the Neiman text for this declaration:

```
#pragma once

#using <mscorlib.dll>          // core functionality
#using <System.dll>            // contains the ImageList class
#using <System.Drawing.dll>    // contains the Size, Icon classes
#using <System.Windows.Forms.dll>    // contains the Button class
using namespace System;
using namespace System::Drawing;
using namespace System::Windows::Forms;

// This is a class declaration that will be used for the Button GUI form.

public __gc class CGui : public Form
{
public:
      // This is to set up the constrcutor for the Windows GUI form
      CGui();
private:
      Button * btnMyButton;
};
```

(Neiman, p. 325). This button class will be used within the GUI of the program for the user to control the simulation, start it, stop it, and perform any basic interactive operations. Now that the class has been declared, I will need to create the Windows form for the button, where the user will interact with the program. This form allows the programmer to set the size and location of the button on the form, along with any text that will be included on the button. In this screen shot, I am creating the button and setting up the color and size:

```
#include "CGui.h"

CGui::CGui()
{
	// This is the title of the Windows form.
	this->Text = S"Black Hole Button";

	// I used black as the background color, to match black holes.
	this->BackColor = Color::Black;


	// Set up the size of your Windows Form:
	// 300 pixels wide and 300 pixels tall
	this->Size = Drawing::Size(300, 300);

	// create Button:
	this->btnMyButton = new Button();

	// Button location:
	this->btnMyButton->Location = Drawing::Point(150, 125);

	// Button size:
	this->btnMyButton->Size = Drawing::Size(100, 50);

	// Button background button:
	this->btnMyButton->BackColor = Color::white;

	// Text on button:
	this->btnMyButton->Text = S"Click Here!";

	// Add to the Windows Form:
	this->Controls->Add(btnMyButton);
}
```

(Neiman, p. 338). This button will be black to match the black hole theme of the project. As the project progresses I will be adding more buttons and modifying the existing ones. The font color and characteristics can also be changed using the following code:

//used to declare the font color within the class:

this->lblMyLabel->ForeColor = Color::White;

//used to set up the font style:

this->lblMyLabel->Font = Drawing::Font(

this->lblMyLabel->Font,

Drawing::FontStyle::Underline

);

(Neiman, p. 382).

The next step will be to create a picture box for the program. This will give me the capabilities to insert an image or series of images into the program (Neiman, p. 471). For this example, I use a JPEG image "Event Horizon". The following code also shows the size of the picture box and where to add it to the Windows form:

```
#include "CGui.h"

CGui::CGui()
{
	this->Text = S"Black Hole Picture Box";

	//Create  PictureBox object:
	this->pbxMyPictureBox = new PictureBox();
	this->pbxMyPictureBox->Location = Point(100,50);
	this->pbxMyPictureBox->Size = Drawing::Size(100, 100);

	//  stretch to fill the area:
	this->pbxMyPictureBox->SizeMode = PictureBoxSizeMode::StretchImage;

	// We can specify the size of the image to be displayed by
	// setting the ClientSize property:
	this->pbxMyPictureBox->ClientSize = Drawing::Size(100, 100);

	// Display the actual image by setting the Image property:
	this->pbxMyPictureBox->Image =
Drawing::Image::FromFile(S"EventHorizon.jpg");

	// Add to the Windows Form:
	this->Controls->Add(this->pbxMyPictureBox);
}
```

Conclusion

The sticking point to this project continues to be converting the Numerical General Relativity Equations into a usable format for Visual C++.NET. As with any black hole simulation, these equations will form the basis of a correct model. I am continuing to research this matter, and to look for similar Numerical Relativity computer models that I can apply to the project.

References

Bakala, P., Ccaron;ermák, P., Hledík, S., Stuchlík, Z., Truparová Plšková, K. (2005). *A virtual trip to the Schwarzschild-de Sitter black hole: computer simulation of extreme gravitational lensing in Schwarzschild-de Sitter spacetimes.* Proceedings of RAGtime 6/7: Workshops on black holes and neutron stars. S. Hledík and Z. Stuchlík, Ed. ISBN 80-7248-334-X. Retrieved September 16, 2009 from *http://adsabs.harvard.edu/full/2005/2005ragt.meet...11B*

Hannam, M. (March, 2009). *Status of black-hole-binary simulations for gravitational-wave detection.* Physics Department, University College Cork, Cork, Ireland. Retrieved September 16, 2009 from http://arxiv.org/PS_cache/arxiv/pdf/0901/0901.2931v2.pdf

Kiritsis, E. (2007). *String theory in a nutshell.* Princeton University Press. ISBN-13: 978-0-691 12230-4.

Neiman, A. (2006). *Hit the ground running with visual C++.NET*. Clark, Ed. Thomson Delmar Learning. ISBN: 1401878806.

Wanjek, C. (February, 2006). *The black hole road show. Goddard helps with first–ever voyage into a black hole.* National Aeronautics and Space Administration 2(3). Retrieved September 16, 2009 from http://www.nasa.gov/centers/goddard/pdf/143246main_Vol2.Issue3-Web.pdf

Creating a Research Program - Part IV

This paper is a continuation of my Visual C++.NET black hole simulation project. This phase involves some of the graphics used in the program and the database access tools I will be using, including a C++ Numerical Relativity data library that provides General Relativity calculations for C++ and actual astronomical data obtained from binary black hole scientific observations.

The Visual C++.NET shapes applet I used for the black hole simulation project. I created an ellipse with a black center and a blue border (event horizon) (Neiman, p. 575). I used the brush for the fill in commands, and the pen to draw the borders.

```
#include "CGui.h"

CGui::CGui()
{
	this->Size = Drawing::Size(400,400);
	this->Text = S"Black hole shape";
	this->Icon = new Drawing::Icon(S"HAMMER.ICO");
	this->BackColor = Color::Black;
};

void CGui::OnPaint(PaintEventArgs * paintEvent)
{
	__super::OnPaint(paintEvent);

	// Draw a circular shape for the radius.
	Graphics * grGraphicsObject = paintEvent->Graphics;

	// Outline of shape:
	Pen * pen = new Pen(Color::BlueViolet, 5);

	// color of brush:
	SolidBrush * sBrush = new SolidBrush(Color::Aqua);


	// the ellipse is the shape of the black hole:
	grGraphicsObject->DrawEllipse(
		pen,
		10, 130,	// upper left corner of rectangle
		200,		// width of rectangle
		50			// height of rectangle
	);

	// fill in the black hole:
	sBrush->Color = Color::Black;
	grGraphicsObject->FillEllipse(
		sBrush,
		10, 130, 200, 50

}
```

This second screen shot is for the Visual C++.NET applet to rotate a black hole ellipse. The ellipse will rotate 15 degrees for 10 additional ellipses (Neiman, p. 580).

```
#include "CGui.h"

CGui::CGui()
{
	this->Size = Drawing::Size(400,300);
	this->Text = S" Rotating black hole";
	this->Icon = new Drawing::Icon(S"ARW05RT.ICO");
	this->BackColor = Color::Black;
};

void CGui::OnPaint(PaintEventArgs * paintEvent)
{
	// refresh
	__super::OnPaint(paintEvent);

	// change the origin and rotate the axes.
	Graphics * grMyGraphicsObject = paintEvent->Graphics;

	Pen * pen = new Pen(Color::Blue, 5);

	//draw first ellipse:
	grMyGraphicsObject->DrawEllipse(pen,0,0,100,50);

	// draw 15 additional ellipses rotated
	// 15 degrees from each other:,
	for (int i=0; i<7; i++)
	{
		grMyGraphicsObject->RotateTransform(15);
		grMyGraphicsObject->DrawEllipse(pen,0,0,100,50);
	}
}
```

These shapes will be used more for demonstration purposes, and not for the actual numerical simulations and calculations. Those will be performed by the actual data base libraries and C++ Numerical Relativity commands. Some Numerical Relativity simulations run in shell domain such as Unix that control the main functions of the program. This shell is part of an external domain system (Jaramillo, 2006). What is crucial to representing a correct simulation model is the interplay between the geometrical, analytical, and numerical components of the simulation. For example, the geometrical aspects include coordinates, and spatial adaptations. The analytical aspect includes equations used to determine the movements and mechanics of the simulation. The numerical aspect includes data collected from astronomical observations (Jaramillo, 2006). These aspects set up the boundary conditions for the simulation run with. My main difficulty is with the analytical aspect of creating the correct Numerical Relativity equations that can be implemented into a Visual C++.Net platform. Most of the existing simulation programs are designed to run on large supercomputer applications, such as a Cray, with programs designed specifically for that computer. However, relief may be in site.

For the database access portion of the project, I used a specific example for black hole simulations, one which will serve the program by providing calculations and an astronomical data library. The example is a C++ Numerical Relativity open source site developed by Greg Cook and Harald Pfeiffer called "Quasi-equilibrium binary black hole initial data" (Cook, Pfeiffer, 2004). This site is consistently updated with new applications and programs. There are essentially two different methods for using the code available in the site, either by a stand alone platform or by accessing one of its code libraries. Examples of the libraries available to developers are: *PublicID.hpp*, the Header file that declares the functions, *libSpEC_ID.a*, the interpolation library for the Header file, and *Kerr-Schild inital data* that supplies the initial data for a three dimensional binary (double) unit-mass spinning black hole (Cook, Pfeiffer, 2004).

The following is the C++ code used to set up and access the public Header files and data library. The *vector* commands are for the coordinates of a black hole in three dimensional (x, y, z) spacetime.

```
#ifndef PublicID_hpp
#define PublicID_hpp

#include <vector>
void InterpolateData(const std::vector<double>& x,
                    const std::vector<double>& y,
                    const std::vector<double>& z,

                    std::vector<double>& gxx,
                    std::vector<double>& gxy,
                    std::vector<double>& gxz,
                    std::vector<double>& gyy,
                    std::vector<double>& gyz,
                    std::vector<double>& gzz,
                    std::vector<double>& Kxx,
                    std::vector<double>& Kxy,
                    std::vector<double>& Kxz,
                    std::vector<double>& Kyy,
                    std::vector<double>& Kyz,
                    std::vector<double>& Kzz,

                    std::vector<double>& Betax,
                    std::vector<double>& Betay,
                    std::vector<double>& Betaz,

                    std::vector<double>& N);
```

This second section of C++ code is used to set up the vector coordinates in the main body of the program, access the data library data base, and perform spin calculations on the vector coordinates, which are the *Kerr-Schild* error calculations at the end. These are used to compare values between those in the data library and what are being calculated in the program (Cook, Pfeiffer, 2004).

```
/// This releases the data being stored in memory
void ReleaseData();

#include <iostream>
#include <vector>
#include <cmath>

#include "PublicID.hpp"

// see any C++ book for explanations of 'using' directive.
using std::vector;
using std::cout;
using std::endl;

int main() {

  // Import data from the current directory on disk, store in RAM.
  // The 'double' parameter represents the orbital angular frequency for
  // the current dataset. (only applicable to BBH initial data)
  ReadData(0);

  // assemble the coordinates of the points to interpolate to.
  const unsigned int N=4; // four points for today

  // (see any C++ book for usage of vector).
  vector<double> x(N),y(N),z(N);
```

```
 x[0]=4;    y[0]=0;   z[0]= 0;
 x[1]=5;    y[1]=0.6; z[1]=-0.6;
 x[2]=1e4;  y[2]=3e4; z[2]= 4e4;
 x[3]=0.5;  y[3]=1.6; z[3]= 0.7;

 // variables returning the interpolated data
 vector<double> gxx, gxy, gxz, gyy, gyz, gzz;
 vector<double> Kxx, Kxy, Kxz, Kyy, Kyz, Kzz;
 vector<double> Shiftx, Shifty, Shiftz, Lapse;

 // Interpolate!
 InterpolateData(x,y,z,
                 gxx, gxy, gxz, gyy, gyz, gzz,
                 Kxx, Kxy, Kxz, Kyy, Kyz, Kzz,
                 Shiftx, Shifty, Shiftz, Lapse);

 for(unsigned int i=0; i<N; ++i) {
   // (see any C++ book for usage of cout and endl).
   cout << "At (" << x[i] << ", " << y[i] << ", " << z[i] << "):" << endl;
   cout << "gxx=" << gxx[i] << ", Kyz=" << Kyz[i]
        << ", beta^y=" << Shifty[i]
        << ", N=" << Lapse[i] << endl << endl;
 };

 // now print some errors
 for(unsigned int i=0; i<x.size(); ++i) {
   const double X=x[i];
   const double Y=y[i];
   const double Z=z[i];
   const double R=sqrt(X*X+Y*Y+Z*Z);
   const double gxx_analytic=1 + 2*X*X/(R*R*R);
   const double gxy_analytic=    2*X*Y/(R*R*R);
   const double N_analytic = 1./sqrt(1+2/R);

   cout << "Difference to the analytical values for Kerr-Schild at ("
        << X << ", " << Y << ", " << Z << "):"  << endl;
   cout << "gxx-gxx_analytic = " << gxx[i] - gxx_analytic << endl;
   cout << "gxy-gxy_analytic = " << gxy[i] - gxy_analytic << endl;
   cout << "N  -  N_analytic = " << Lapse[i] - N_analytic << endl;
   cout << endl;
 }
};
```
(Cook, Pfeiffer, 2004).

I will need to convert some of this code over to Visual C++.NET and run some tests to verify that it will work correctly in my updated version. The good thing about this site is that it is open source and is constantly being improved and upgraded by various users in the Numerical

Relativity development community. I can join this community and then work with the other developers to help set up any modifications and upgrades to my project, and then share my work to assist others with their projects.

One modeling method used to determine both the numerical and analytical aspects of Numerical Relativity is the "Bubble Spacetime" of Kaluza-Klein Theory (Lehner, Neilsen, Sarbach, Tiglio, 2004). The Kaluza-Klein Theory (KKT) was developed in the early 1900s by the two physicists who have its namesake. It treats the four dimensions (three space, one time) as a cylinder, with a fifth dimension as the length of the cylinder. KKT is a precursor to String Theory and other multi-dimensional models. The basic properties of the theory make it straightforward to incorporate into computational models. By adding another term in the equation, there is an additional dimension. In the bubble version of KKT, the additional dimensions "shrink to zero in a bubble". Some of the equations that the bubble model uses and can be incorporated into a black hole simulation are:

$ds^2 = -dt^2 + U(r)dz^2 + dr^2/U(r) + r^2 d\Omega^2$

Which is the equation defines a change of measurement in four dimensional spacetime with

$d\Omega^2 = d\theta^2 + \sin^2\theta d\varphi^2$ as the change in the measurement metric for a wave on a smooth sphere and where:

$-dt^2$= the change (negative) in the time coordinate

$U(r)$= a positive radius smooth function for a manifold (shape) of the black hole

dz^2= a change in the coordinate in the "z" (up) direction for the period $P=4\pi\ U(r.)$

The electromagnetic field^ equation on this black hole space time is:

$\frac{1}{2} F_{\mu\nu} dx^n \wedge dx^n = d\gamma(r) \wedge dz$

Where $F_{\mu\nu} dx^n$ = the electric force field with the coordinates μ and ν; $n=\mu$

^ = the convergence (combining) of these terms

dx^n = the change in the x coordinate; $n=\nu$

$d\gamma(r)$ = the change in a smooth radius function as r ->º

(Lehner, et. al., 2004).

Conclusion

In conclusion, Microsoft Visual C++.NET graphical interface tools will be an important aspect of my black hole simulation program. The shapes and drawing tools are valuable for the visual portion of the simulation, and can help to add to the attractiveness of the program if done properly.

References

Cook, G., Pfeiffer, H. (2004). *Quasi-equilibrium binary black hole initial data*. PRD 70, 104016. Retrieved October 3, 2009 from http://www.black-holes.org/researchers3.html

Hong, J. I., Landay, J. A., Van Duyne, D. K. (2007). *The design of site: patterns for creating winning web site.* (*2nd ed.).* Prentice Hall publishers. ISBN: 0131345559

Jaramillo, J.L. (May 22, 2006). *Quasi-local black hole horizons in Numerical Relativity: a quasi equilibrium case.* Laboratoire de l'Univers et de ses Th´eories (LUTH) Observatoire de Paris, Meudon, France. Retrieved October 3, 2009 from http://sciviz.cct.lsu.edu/projects/vish

Lehner, L., Neilsen, D. Sarbach, O., Tiglio, M. (December 13, 2004). *Recent analytical and numerical techniques applied to the Einstein equations*. Department of Physics & Astronomy, Louisiana State University, Baton Rouge, LA, Department of Physics & Astronomy, Brigham

Young University, Provo, UT, California Institute of Technology, Pasadena, CA, Center for Computation and Technology, Louisiana State University, Baton Rouge, LA, Center for Radiophysics and Space Research, Cornell University, Ithaca, NY. Retrieved October 3, 2009 from http://arxiv.org//PS_cache/gr-qc/pdf/0412/0412062v1.pdf

Neiman, A. (2006). *Hit the ground running with visual C++.NET*. Clark, Ed. Thomson Delmar Learning. ISBN: 1401878806