# SIMULACIONES FÍSICAS IMPLEMENTANDO PYTHON Y BLENDER

# PHYSICAL SIMULATIONS IMPLEMENTING PYTHON AND BLENDER

# LIQUIDIFICADOR E SIMULAÇÕES PYTHON E BLENDER


**Hernán Porras[a], Oscar Barrios[d], Andrés Jiménez López[b], Jesús Arias Hernández[c]**

[a] Facultad Ciencias Básicas e Ingeniería, Grupo de Investigación Sistemas Dinámicos, Universidad de los Llanos, **hernan.porras@unillanos.edu.co**

[b] Facultad Ciencias Básicas e Ingeniería, Grupo de Investigación Sistemas Dinámicos, Universidad de los Llanos, **ajimenez@unillanos.edu.co**

[c] Facultad Ciencias Básicas e Ingeniería, Grupo de Investigación Sistemas Dinámicos, Universidad de los Llanos, **jariar@unillanos.edu.co**

[d] Facultad Ciencias Básicas e Ingeniería, Grupo de Investigación Sistemas Dinámicos, Universidad de los Llanos, **oscar.barrios@unillanos.edu.co**



Fecha de recepción: 25 de julio 2017

Fecha de aprobación: 02 de febrero 2018

## Resumen

Este artículo presenta los resultados obtenidos en el proyecto de investigación denominado: Simulaciones de mecánica clásica con la implementación de software libre, como herramienta para el aprendizaje de física mecánica en los estudiantes de ingeniería electrónica y de sistemas de la Universidad de los Llanos. Se explica un método para la comunicación externa de Python y Blender Game Engine mediante Socket, que corresponde a un método para el intercambio de flujo de datos entre dos programas, de forma tal que se pueda controlar un objeto en Blender Game Engine con un código externo de Python, otorgando como resultado la simulación en 3D de algunos fenómenos de física clásica. Este es un resultado de investigación del grupo de Sistemas Dinámicos de la Universidad de los Llanos.


**Palabras clave:** Blender, Game Engine, Game Logic, Multiplataforma, Python, Simulación.


## Abstract.

This article present the results obtained in the denominated project of investigation: Simulations of classic mechanics with the implementation of free software, like tool for the learning of mechanical physics in the students of electronic engineering and systems of the University of the Llanos. A method for the external communication of Python and Blender is explained Engine Game by means of Socket, that corresponds to a method for the interchange of data flow between two programs, so that an object in Blender can be controlled Engine Game with an external code of Python, granting like result the simulation in 3D of some phenomena of classic physics. This it is a result of investigation of the group of Dynamic Systems of the University of the Llanos.


**Key words:** Blender, Engine Game, Logic Game, Cross-platform, Python, Simulation.


## Resumo

Este artigo apresenta os resultados do projeto de pesquisa chamado: Simulações de mecânica clássica com a implementação do software livre, como uma ferramenta para aprender mecânica física estudante de engenharia eletrônica e sistemas da Universidade de Ihanos. Explica-se um método para a comunicação externa Pitão e de Blender Gama Ferramenta por Socket, que corresponde um método para a troca de fluxos de dados entre dois programas, de modo que possa-se controlar um objeto em Blender Gama Ferramenta com um código externo de Pitão, como resultado a simulação 3D de cerca de fenómenos de física clássica. Este é o resultado de investigação do grupo de Sistemas Dinâmicos da Universidade de que é de nível.


**Palavras chave:** Blender, Gama Ferramenta, Gama Logic, Multiplataforma, Pitão, Simulação.

# I. INTRODUCCIÓN

La simulación es una disciplina en auge en el ámbito de la ingeniería, debido a los avances en la generación de gráficos por ordenador y a la aparición de nuevos programas que permiten crear y visualizar dichos gráficos de forma cada vez más sencilla. Las simulaciones tienen gran importancia en áreas educativas, la cual sirve de herramienta a los estudiantes para aumentar el grado de asimilación del conocimiento en áreas científicas como la física clásica, que para algunos es difícil de entender(Rojas, Morales, Rangel, & Torres, 2009)(Daniel et al., 2016)(Morales, 2010).

Con el avance de la tecnología y el uso de lenguajes de programación como Python el cual es alto nivel lo cual ofrece una buena manera de expresar una idea (Rojas, Martínez, & Morales, 2014), además de sus características son muy útiles en cualquier área de desarrollo, este tiene la facilidad de colaborar con una amplia variedad de lenguajes, permitiéndole al programador ser más productivo y hacer de Python una herramienta de desarrollo muy atractiva.(Martelli, 2008)

Por otra parte, Blender es un recurso de software libre que permite la posibilidad de disponer de herramientas de diseño profesional 3D. Su código fuente está disponible al público, permitiendo la integración de un intérprete de lenguaje para que usuarios menos técnicos puedan aprovechar todo su potencial. Blender escogió a Python, como lenguaje script, permitiendo la realización de aplicaciones vistosas y con posibilidad de interpretar datos externos en un video juego o un entorno interactivo por medio de una conexión (Ruiz, 2008).

Blender cuenta con paquetes de creación totalmente integrados, ofreciendo un amplio rango de herramientas esenciales para la creación de contenido 3D, incluyendo: modelado, mapeo de texturas o UV, texturizado, construcción de restricciones para objetos, ejemplo los huesos humanos, animación, simulación de partículas y otros, interprete de Python embebido, renderizado, composición, post-producción y creación de juegos. También es un software multiplataforma que cuenta con una interfaz unificada basada en OpenGL compatible para los sistemas operativos más comunes: Windows (98, NT, 2000, XP, Vista, 7 y 8), Linux, MAC OSX (Tiger, Leopard, Snow Leopard, Lion y Mountain Lion), FreeBSD, Irix y Sun (Aguila, 2013).

El desarrollo de videojuegos basados en la tridimensionalidad es uno de los mayores avances en su historia desde un punto de vista visual y un cambio de paradigma con las lógicas de juego que imponían las dos dimensiones. El uso de una de estas dos posibilidades de representación espacial a la hora de desarrollar un videojuego es una decisión que implica consecuencias de cómo se desarrollará gráficamente y cómo el usuario interactuará con él (Bacone, 2012).

Blender dispone de un motor de video juegos (Blender Game Engine), el cual se puede usar para crear video juegos online por medio de protocolos de comunicación. Para facilitar el uso externo de Python con Blender, la comunicación se realiza por medio de sockets, que son una extensión del sistema operativo, que entablan comunicación con procesos, programas y máquinas.

En los sistemas operativos los procesos se clasifican en dos grupos, procesos cooperativos o independientes, para este caso los procesos que se presentan son independientes (Gharehchopogh, Amini, & Maleki, 2014); la comunicación entre procesos o IPC (Inter Process Communication, por sus siglas en inglés), estás herramientas provee procesos para tener una interacción sin necesidad de dependencia. Estas herramientas de bajo nivel

permite el intercambio de mensajes y primitivas entre procesos, un ejemplo de esto es acceso directo a la API (Interfaz de programación de aplicaciones) de los protocolos de internet o programación de sockets, los cuales usan distintos protocolos de comunicación.(Ghate & Pati, 2016).

De las diferentes formas de IPC, los sockets son de lejos la más popular para comunicaciones interplataforma (Werther, 2013).

Los sockets como parte del IPC, pueden ser usados en cualquier lenguaje de programación sin importar el sistema operativo; es un enchufe o punto virtual donde diferentes procesos pueden comunicarse estando en diferentes computadores (Sarker & Washington, 2015), los sockets usan como protocolos de comunicación a TCP y UDP, los cuales se encuentran en la capa de transporte en el modelo TCP/IP, la interacción entre sockets se hace Cliente/Servidor, donde el Servidor siempre se presta a esperar y escuchar Clientes, mientras que las peticiones de servicio se realiza al Servidor, otorgando cómo resultado una respuesta de acuerdo a la solicitud; dependiendo la aplicación en la cual se implementa los sockets, se puede usar TCP o UDP, TCP presta un servicio orientado a conexión verificando la entrega de datos entre el cliente y el servidor, al cambio UDP presta un servicio no orientado a conexión y su priorización es solo enviar datos sin importar la verificación de los mismos.(Welch & Jones, 2014).

Los resultados mostrados en este documento corresponden a los obtenidos por el grupo de investigación de Sistemas Dinámicos de la Universidad de los Llanos en el marco del proyecto de investigación FCBI-13-2013.

## II. DESARROLLO DE LA SIMULACIÓN

Este proyecto se realiza con una Workstation de 32 Gb de RAM, procesador Intel Xeon, tarjeta de video AMD FirePRO W5100 y Ubuntu 14.04 LTS (Precise Pangolin).

El proceso de desarrollo se ha realizado en dos etapas, una etapa que consta del desarrollo de un programa de modelado numérico en Python con entorno gráfico y otra etapa en el desarrollo de la simulación en Blender con los datos obtenidos de Python y la conexión con Python.

Se usan los Sockets para él envió y recepción de información, mientras Python envía la información a Blender y este la interpreta en coordenadas “X-Y-Z” para el objeto o la figura correspondiente, con Game Logic se crea los scripts de Python para que el objeto seleccionado reciba los datos del Socket externamente, método desarrollado:

### *A. Desarrollo en Python*

*1)*

El lenguaje de programación Python es conocido por su gran potencia en investigación y cálculos matemáticos, además de ser un lenguaje de Alto nivel y orientado a objetos.

Por otra parte, como todo lenguaje de programación se realiza importación de librerías (ver Figura 1), estas librerías que se usan son para interactuar con funciones propias del lenguaje de programación, para importar una librería se utiliza la palabra reservada import o si se necesita interactuar directamente con un paquete de la librería se utiliza from, además puede importar

todos los paquetes de la librería; también se le puede agregar acrónimos o palabras más cortas para poder realizar un llamado a la librería, es usado para hacer referencia a la librería y no escribir el nombre completo de esta. En el desarrollo del proyecto se usó las librerías sys, matplotlib, pyplot, numpy, math, socket, QtCore, PyQt4, QtGui, entre otras.

```
1.  # -*- coding: utf-8 -*-
2.  import sys
3.  import matplotlib.pyplot as plt
4.  import numpy as np
5.  import math
6.  import socket
7.  import time
8.  from PyQt4 import QtCore, QtGui
9.  from PyQt4.QtCore import *
10. from PyQt4.QtGui import *
11. from interfaz_3 import Ui_Dialog
```

**Figura 1.** Librerías usadas para el desarrollo del proyecto.
Fuente: Autor del proyecto.

Para poder generar ventanas se utiliza la ayuda de la herramienta de Qt4 designer (es la ayuda para generar ventanas gráficas, ya que Python no tiene su propio motor de interfaces).

Como se sabe en cualquier lenguaje de programación se utiliza un método o función principal llamado main para ejecutar toda la lógica del software (ver Figura 2), en el main se hace los llamados a las funciones que generan la interfaz y mostrar.

```
12. if __name__=="__main__":
13.     app = QtGui.QApplication(sys.argv)
14.     ventana = ventanaSimulacion()
15.     ventana.show()
16.     sys.exit(app.exec_())
```

**Figura 2.** El main principal para la ejecución del software.
Fuente: Autor del proyecto.

Las funciones en Python se declaran con la palabra def y en el caso de ventanas graficas se usa __init__ la cual es para indicar el inicio el llamado de los métodos pintar y organizar la ventana.

```
17. def __init__(self):
18.   QtGui.QWidget.__init__(self)
19.   self.ventana=  Ui_Dialog()
20.   self.ventana.setupUi(self)                    './I
21.   self.ventana.imagen.setPixmap(QtGui.QPixmap(
    magenes/unillanos2.png'))
22.   self.connect(self.ventana.simular, QtCore.SIGNAL
    ('clicked()'), self.simular)
23.   self.connect(self.ventana.caida_libre, QtCore.SI
    GNAL('clicked()'), self.activar_campos)          Qt
24.   self.connect(self.ventana.rectilinioUniforme,
    Core.SIGNAL('clicked()'), self.activar_campos)
25.   self.connect(self.ventana.rectilinioUniformement
    eAcelerado, QtCore.SIGNAL('clicked()'), self.activar
    _campos)                                        QtCore.
26.   self.connect(self.ventana.movParabolico,
    SIGNAL('clicked()'), self.activar_campos)
27.   self.activar_campos()
```

**Figura 3.** Código de creación de la ventana y sus respectivas acciones sobre ellas.
Fuente: Autor del proyecto.

Seguidamente, se puede identificar como se generan todos los valores de los movimientos o dicho de otra manera, la función que describe el movimiento de cada uno de los fenómenos físicos (ver Figura 4), teniendo en cuenta que solo usamos una ecuación para poder hacer cualquiera de los movimientos en su respectivo eje de coordenada cartesiana, recibiendo valores como lo son: ángulo, aceleración, tiempo, posición inicial y velocidad inicial (se tiene que tener en cuenta que estas funciones que se generan son para la posición ya sea en el eje x o eje y); al ingresar determinados valores en la ecuación realiza el movimiento respectivo.

```
28.   def ecuacion_principal_x(self,xi,vi,thetha,ax,t):

29.     return (xi+((vi*math.cos(thetha))*t)+((ax*(t**
    2))/2))
30.
31.   def ecuacion_principal_y(self,yi,vi,thetha,ay,t):

32.     return (yi+((vi*math.sin(thetha))*t)+((ay*(t**
    2))/2))
```

**Figura 4.** Ecuaciones de movimiento en X y Y.
Fuente: Autor del proyecto.

La forma de validar los datos faltantes se encuentra en la Figura 5, la forma de identificar qué tipo de error tiene para no poder realizar el cálculo es de preguntar si algún campo está vacío, después se procede a preguntar si cada uno de los

datos son numéricos y no se encuentra una letra entre ellas o carácter especial, por último, se pregunta si los datos ingresados no superan el límite permitido.

```
33.  elif self.ventana.rectilinioUniforme.isChecked()
    :
34.      print "esta seleccionado rectilinio Uniforme
    "
35.      if (self.ventana.velocidad.text()=="") or (s
     elf.ventana.aceleracion.text()==""):
36.              QtGui.QMessageBox.information(self, "Inf
     ormacion", "Lo sentimos, algunos campos estan vacios
     ", QtGui.QMessageBox.Ok)
37.      elif (self.comprobar(self.ventana.velocidad.
     text())==True):
38.              QtGui.QMessageBox.information(self, "Inf
     ormacion", "Lo sentimos, has ingresado letras o un c
     aracter especial", QtGui.QMessageBox.Ok)
39.          elif (self.comprobar(self.ventana.aceleracio
     n.text())==True):
40.              QtGui.QMessageBox.information(self, "Inf
     ormacion", "Lo sentimos, has ingresado letras o un c
     aracter especial", QtGui.QMessageBox.Ok)
41.          elif (float(self.ventana.velocidad.text())<1
     00) and (float(self.ventana.velocidad.text())> -
     100):
42.              self.abrir_puerto(2)
43.          else:
44.              QtGui.QMessageBox.information(self, "Inf
     ormacion", "Los valores estan excediendo los permitd
     os, la velocidad debe ser menor que 100  y mayor a -
     100", QtGui.QMessageBox.Ok)
```

**Figura 5.** Ejemplo de validación de los campos de captura de datos.
Fuente: Autor del proyecto.

El algoritmo generado para realizar el cálculo de los datos se basa en recoger la información de cada campo ingresado (ver Figura 6), se prosigue a definir las constantes para el movimiento seleccionado y los valores que cambian con el tiempo; se tiene que saber que los intervalos de tiempo que se están generando o mejor la variable dt es para poder observar la simulación y se note el trayecto; se genera una información de fecha y hora para guardar múltiples archivos .xls (se hablara más claramente de este más adelante) y su gráfica. Se crea el archivo donde guarda los datos de la simulación, se prosigue a usar un ciclo el cual se termina hasta el límite decidido por el usuario, el cual fue ingresado en el campo tiempo.

Dentro del ciclo se realiza el llamado de la función, la cual retorna la posición, además de generar la gráfica o graficas de los valores una vez terminado el proceso del movimiento de física seleccionada.

```
45. v0=float(self.ventana.velocidad.text())
46.             tiempo=float(self.ventana.aceleracion.tex
    t())
47.             seconds=0
48.             contador=0
49.             gravity=9.8
50.             dt=0.0001 #dt is delta t and is 0.1 secon
    d
51.             finished=False
52.             info=time.strftime("%b")
53.             info+="-"+time.strftime("%d")
54.             info+="-"+time.strftime("%a")
55.             info+="-"+time.strftime("%Y")
56.             info+="-"+time.strftime("%H")
57.             info+="-"+time.strftime("%M")
58.             info+="-"+time.strftime("%S")
59.             f=open("./Grafias  y Resultados Rectilinio
    /datosMRU-"+str(info)+".xlsx",'a')
60.             f.write( "PosiciÃ³n:;Tiempo:\n")
61.             while not finished:
62.             seconds +=dt
63.             posX=self.ecuacion_principal_x(0.0,(
    v0+0.0),0.0,0.0,seconds)
64.             f.write(str(posX)+";"+str(seconds)+"
    \n")
65.             pos_x_graf.append(posX)
66.             tiem_graf.append(seconds)
67.             self.envio(posY,0,posX)
68.             if seconds>=tiempo:
69.               finished=True
70.             plt.plot(tiem_graf,pos_x_graf,line
    style='dashdot', linewidth=4,marker='o',markerfaceco
    lor='green',markeredgecolor='black',markeredgewidth=
    0.01,markersize=0.01)
71.             plt.xlabel('tiempo (s)')
72.             plt.ylabel('Posicion (m)')
73.             plt.title('Posicion Vs tiempo')
74.             plt.grid(True)
75.             plt.savefig( './Grafias  y Resultado
    s Rectilinio/'+str(info)+".png")
76.             plt.show()
77.             f.close()
78.             print 'seconds to drop: ' +str(sec
    onds)
79.             self.ventana.tabla_datos.insertRow(c
    ontador)
80.             self.ventana.tabla_datos.setItem(con
    tador,0,QTableWidgetItem(str(posX)))
81.             self.ventana.tabla_datos.setItem(con
    tador,4,QTableWidgetItem(str(seconds)))
82.             contador+=1
```

**Figura 6.** Código de la lógica de generar los valores de la simulación.
Fuente autor.

Para terminar la parte de desarrollo en Python se realizó una función de cambio de etiquetas (labels o label) (ver Figura 7), la cual consiste en cambiar el nombre del label e indicar que tipo de campos debe mostrar la ventana; lo cual dependiendo del tipo de movimiento se activaran o desactivaran campos.

```
83.  if self.ventana.caida_libre.isChecked():
84.      self.ventana.labAcel.setText("Altura")
85.      self.ventana.lab2.setText("m")
86.      self.ventana.aceleracion.show()
87.      self.ventana.labTheta.hide()
88.      self.ventana.angulo.hide()
89.      self.ventana.lab3.hide()
90.      self.ventana.lab2.show()
91.      self.ventana.labAcel.show()
```

**Figura 7.** Ejemplo de activación de casillas y labels.
Fuente: Autor del proyecto.

### *B. Desarrollo en Blender 2)*

Se inicia con la conexión básica de Sockets la cual se divide en dos partes "Cliente y Servidor". Esto se hace en dos scripts, donde se definen variables generales como un numero de puerto común por lo cual el Cliente y el Servidor se comunican, culminado este proceso y verificado la comunicación de ambos scripts se procede a usar Blender con el BGE, donde se creará un objeto que se moverá con los datos que reciba de la comunicación con el script externo de Python. Luego usando la interfaz Game Logic donde al objeto se le adicionara un sensor, controlador y un actuador, junto con un script de Python interno que se crea para recibir los datos enviados externamente. El Cliente será el script externo de Python y el Servidor será Blender.

Para la comunicación básica de un socket entre un Cliente y un Servidor se deben tener en cuenta dos variables importantes, el puerto y la dirección IP para la comunicación entre ellos, los siguientes scripts presentados son modelos básicos de Cliente y Servidor.

En el proyecto se inició con el diseño de los scripts Cliente-Servidor en Python, para hacer su implementación con Blender y así controlar objetos o modelos para la simulación externa, se opta por usar UDP como protocolo de comunicación para los sockets, puesto que no verifica los paquetes que se envían, él envió información es más rápido respecto al TCP.

En las Figura 8 y Figura 9 se observan los códigos bases para el cliente y el servidor de la aplicación, en este caso el cliente será el programa Python que genera los datos de la simulación y Blender será el servidor que interpretará estos datos en movimiento para un objeto predefinido.

```
92. import socket
93.
94. UDP_IP="127.0.0.1"
95. UDP_PORT=52248
96. z=-2
97. MESSAGEz=str(z)
98. print("coordenadas Z: ",z)
99. #Envio de datos al servidor
100.sock=socket.socket(socket.AF_INET,  socket.SOCK_DGRAM
    )
101.sock.sendto(MESSAGEz,(UDP_IP, UDP_PORT))
102.sock.close()
```

**Figura 8.** Código UDP Python para el cliente.
Fuente: Autor del proyecto.

```
103.#servidor
104.import socket, sys
105.
106.UDP_IP="127.0.0.1"
107.UDP_PORT=52248
108.sock=socket.socket(socket.AF_INET,  socket.SOCK_DGRAM
    )
109.sock.bind((UDP_IP, UDP_PORT))
110.#Envio de datos al servidor
111.try:
112.    while True:
113.        dataZ, addrz=sock.recvfrom(1024)
114.        print("Dato en dataZ: ",str(dataZ))
115.finally :
116.    sock.close()
```

**Figura 9.** Código UDP Python para el servidor.
Fuente: Autor del proyecto.

Con base a los códigos mostrados anteriormente, el servidor cambia para Blender, su intérprete de Python el cual esta embebido maneja una estructura de ejecución de código distinta, además que se agrega código propio de Blender para suministrar movimiento al ver Figura 10.

```
117. import bge, socket
118. from random import random
119.
120. def move():
121.     cont=bge.logic.getCurrentController()
122.     own=cont.owner
123.     scene=bge.logic.getCurrentScene()
124.     UDP_IP="127.0.0.1"
125.     UDP_PORT=52248
126.      sock=socket.socket(socket.AF_INET, socket.SOCK_D
     GRAM)
127.     sock.bind((UDP_IP, UDP_PORT))
128.     try:
129.        #Recepción de datos
130.        dataZ, addrz=sock.recvfrom(1024)
131.        print("Posicion Z: " float(dataZ))
132.        #Asignación de datos al objeto para su movimi
     ento
133.        own.position.z=float(dataZ)
134.     finally :
135.        sock.close()
136.
137. move()
```

**Figura 10.** Código UDP Python par Blender.
Fuente: Autor del proyecto.

El código del cliente también se modifica para enviar a Blender las respectivas coordenadas al objeto para el movimiento.

```
138.  def envio(self,y, x , z):
139.     UDP_IP = "127.0.0.1"
140.     UDP_PORT = 52248
141.     MESSAGEz = str(z)
142.     MESSAGEy = str(y)
143.      sock = socket.socket( socket.AF_INET, socket.S
     OCK_DGRAM)
144.     sock.sendto( MESSAGEz, (UDP_IP, UDP_PORT))
145.     sock.sendto( MESSAGEy, (UDP_IP, UDP_PORT))
```

**Figura 11.** Cliente UDP Python modificado para envió de datos a Blender.
Fuente: Autor del proyecto.

Definida la comunicación se procede a realizar un ejemplo de la aplicación para mostrar el funcionamiento básico; lo primero es crear un objeto, que puede ser: un cubo, una esfera, entre otros y cambiar el motor de procesamiento interno a procesamiento de juego, ver Figura 12.

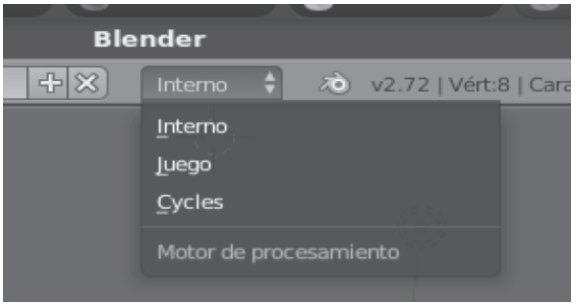


**Figura 12.** Cambio a motor de video juegos de Blender.
Fuente: Autor del proyecto.

A continuación, en GameLogic se agrega un script tipo Python, el cual contiene código del server y continuo a ello se hacen las configuraciones correspondientes en el editor de lógica y el script para su funcionamiento para recibir la información de movimiento para el objeto (Ver Figura 13).

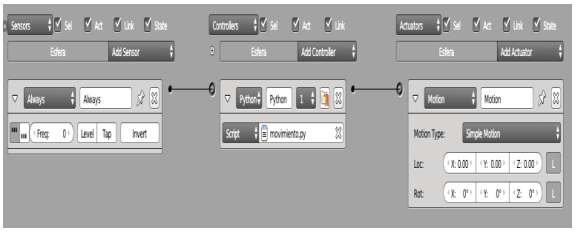

**Figura 13.** Configuraciones para ejecución del script Python en Blender.
Fuente: Autor del proyecto.

Ahora la aplicación en Blender está lista para recibir la información para otorgar movimiento a un objeto.

## III. RESULTADOS

### A. *Resultados en Python*

*3)*

El software tiene una interfaz donde puede encontrar diferentes tipos de movimientos:

- Caída Libre.
- Movimiento Rectilíneo Uniforme.
- Movimiento Rectilíneo Uniformemente Variado.
- Movimiento Parabólico.

En la interfaz se encuentran campos donde se ingresa datos (ver Fig. 14), una ventana de selección de la simulación a realizar, una tabla de resultados y el botón de simular.


**Figura 14.** Interfaz de ingreso de datos.
Fuente: Autor del proyecto.

Se comienza por conocer los campos en los cuales se ingresaran los datos, estos campos para cada una de las simulaciones son diferentes, es decir, en Caída Libre los campos a ingresar son velocidad y altura, en Movimiento Rectilíneo Uniforme es velocidad y tiempo (el tiempo que ingresa es el tiempo de duración de la simulación), en Movimiento Rectilíneo Uniformemente Variado es velocidad, aceleración y tiempo (el tiempo que ingresa es el tiempo de duración de la simulación) y el Movimiento Parabólico los campos son velocidad y theta (theta es el ángulo de lanzamiento del objeto). Para activar estos campos primero tiene que seleccionar el tipo de movimiento (ver Figura 15 y Figura 16).

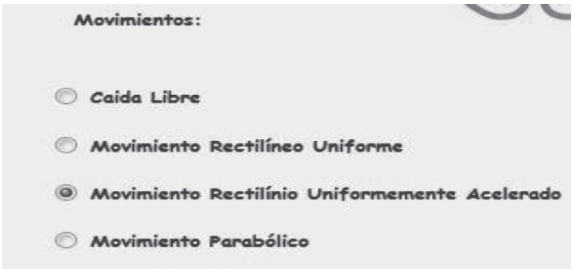


**Figura 15.** Tipos de movimiento.
Fuente: Autor del proyecto.


**Figura 16.** Ingreso de datos movimiento rectilíneo Uniformemente Acelerado.
Fuente: Autor del proyecto.

Ahora en la tabla de resultados se pueden observar los resultados de las simulaciones (ver Fig. 17), es decir, se encuentran los datos numéricos generados por el movimiento experimental. La información que se encuentra es la posición, la velocidad y el tiempo, además el tiempo que muestra es el tiempo trascurrido desde el inicio de simulación hasta su finalización.

| | Posición X: | Posición Y: | Velocidad X: | Velocidad Y: | Tiempo: |
|---|---|---|---|---|---|
| 23765 | 103.298522678 | 0.00476970686945 | 43.466662183 | -11.6428429704 | 2.3765 |
| 23766 | 103.302869344 | 0.0036053726724 | 43.466662183 | -11.6438229704 | 2.3766 |
| 23767 | 103.30721601 | 0.00244094127536 | 43.466662183 | -11.6448029704 | 2.3767 |
| 23768 | 103.311562677 | 0.00127641197832 | 43.466662183 | -11.6457829704 | 2.3768 |
| 23769 | 103.315909343 | 0.000111784681273 | 43.466662183 | -11.6467629704 | 2.3769 |

**Figura 17.** Ejemplo de datos mostrados del movimiento parabólico.
Fuente: Autor del proyecto.

Al finalizar el cálculo de los valores, se genera una o varias gráficas (depende del tipo de movimiento realizado se muestra las gráficas ver Figura 18 y Figura 19), se creara un archivo .xls o mejor conocido como Excel, este Excel generado contiene los datos cada instante de tiempo mientras la simulación se encuentra en ejecución, esto se encuentra en un carpeta que contiene los resultados mostrados en la tabla, además guarda las gráficas mostradas (cabe recalcar que para cada movimiento tiene su respectiva carpeta).

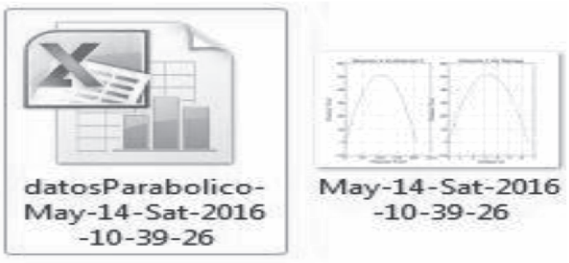


**Figura 18.** Ejemplo de los archivos guardados tras los cálculos de un movimiento.
Fuente: Autor del proyecto.

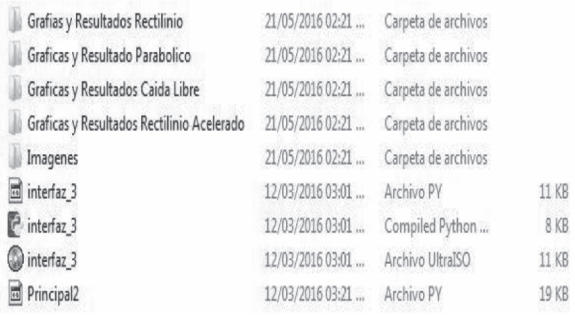


**Figura 19.** Ejemplo de las carpetas de las simulaciones y los archivos de ejecución de la simulación.
Fuente: Autor del proyecto.

Uno de los posibles errores es al ingreso de datos (ver Figura 20 y Figura 21), por ejemplo, el campo de ingreso de datos no tiene permitido ingresar valores muy grandes o mayores a 99, también al no ingresar todos los datos para la simulación se generan otro error.

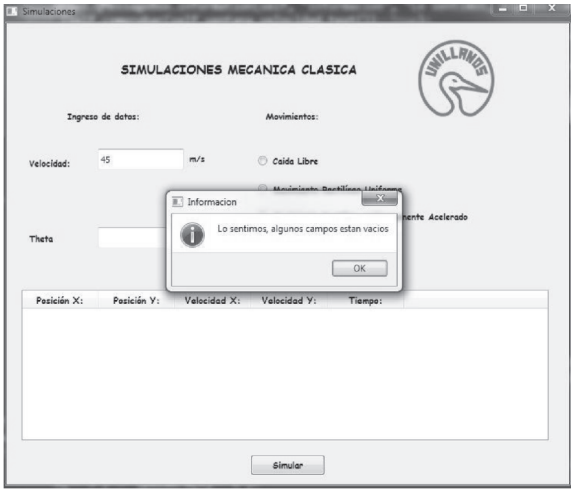


**Figura 20.** Ejemplo que muestra la falta de un campo.
Fuente: Autor del proyecto.


**Figura 21.** Ejemplo excediendo un valor no permitido.
Fuente: Autor del proyecto.

Finalmente se conoce la interfaz a trabajar y sus respectivas limitaciones, un ejemplo de la aplicación se puede ver en la Figura 22, la cual se ingresa una velocidad de 45 metros sobre segundo y un ángulo de 45 grados, además muestra sus respectivas gráficas autogeneradas por el tipo de movimiento físico.

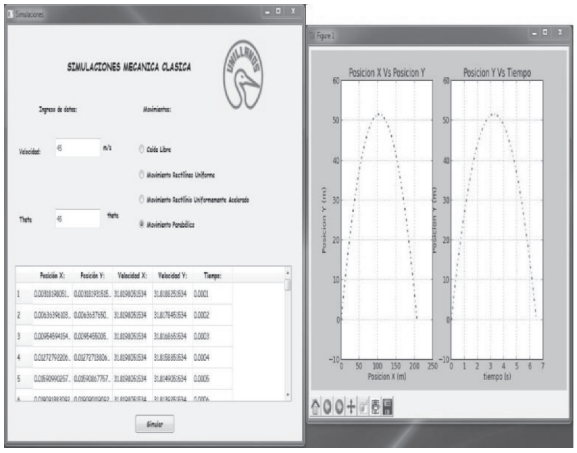


**Figura 22.** Ejemplo del movimiento parabólico.
Fuente: Autor del proyecto.

## B. *RESULTADOS BLENDER*

Como se observa en la Fig. 22 la esfera se mueve con respecto a los datos generados en el cliente el cual se encuentra a la izquierda de Blender, en el desarrollo del código servidor para Blender.

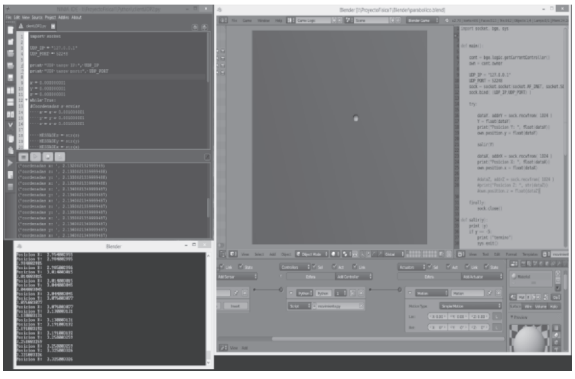

**Figura 23.** Movimiento de una esfera en 2 dimensiones "Ejemplo".
Fuente: Autor del proyecto.

Para la realización de las simulaciones en Blender se crean escenarios con objetos prediseñados libres para dar una mejor ambientación visual, mejorando la experiencia del usuario con la simulación. En la Fig. 23. Se muestra como ejemplo una de las simulaciones diseñadas en Blender.

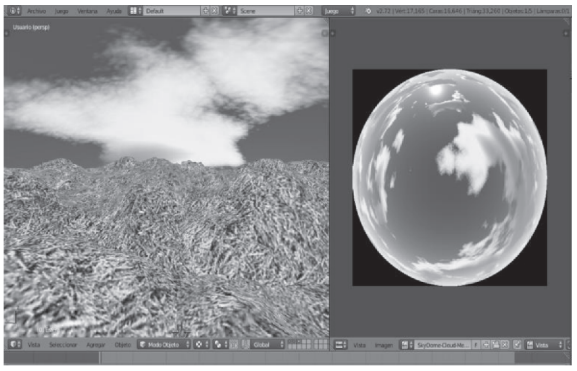

**Figura 24.** Escenario para simulación con un terreno prediseñado en forma de montaña.
Fuente: Autor del proyecto.

En la Figura 24 y Figura 25, se han agregado un terreno en forma de montaña y un skydome para simular el cielo en un día normal.

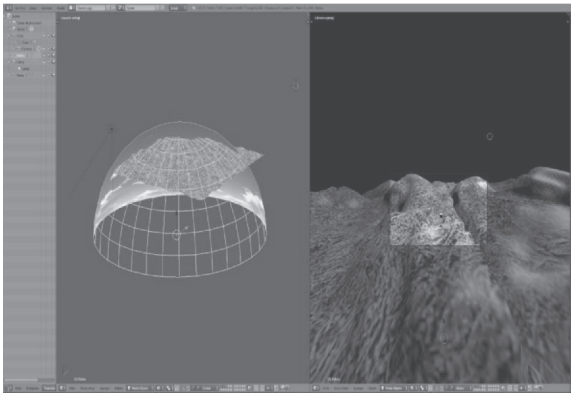

**Figura 25.** Skydome con el terreno.
Fuente: Autor del proyecto.

Se tiene otra perspectiva de vista de cómo se organiza un objeto en Blender; en la parte derecha se tiene una prevista de la cámara hacia el terreno, a continuación, se observa el objeto a manipular, el cual es un cubo. Ver Figura 26.

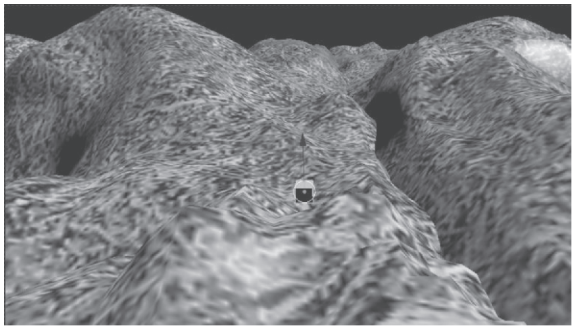

**Figura 26.** Vista desde la cámara asignada al cubo.
Fuente: Autor del proyecto.

Para una función optima de las simulaciones, en el caso de Movimiento Rectilíneo Uniforme o Movimiento Rectilíneo Uniformemente Acelerado, al script del servidor en Blender se agregan líneas de código Blender Game Engine "BGM" como resultado un terreno infinito, este terreno funciona recursivamente. Ver Fig. 27.

```
145.import bge, socket
146.from random import random
147.
148.def move():
149.    cont=bge.logic.getCurrentController()
150.    own=cont.owner
151.    scene=bge.logic.getCurrentScene()
152.    UDP_IP="127.0.0.1"
153.    UDP_PORT=52248
154.     sock=socket.socket(socket.AF_INET, socket.SOCK_D
    GRAM)
155.    sock.bind((UDP_IP, UDP_PORT))
156.    try:
157.       #Recepción de datos
158.       dataZ, addrz=sock.recvfrom(1024)
159.       dataY, addrY=sock.recvfrom(1024)
160.       if(float(dataY)  != -1):
161.            #Asignación de datos al objeto para su mo
    vimiento
162.           own.position.z=float(dataZ)
163.     else:
164.            scene.addObject("Plane","Cube",100).world
    Position=[10,float(dataZ) -10,-5]
165.            scene.addObject("Plane","Cube",100).world
    Position=[10,float(dataZ) -15,-5]
166.            scene.addObject("Esfera","Cube",100).worl
    dPosition=[10,float(dataZ)+ 10,-5]
167.    if(float(dataY)  == -2):
168.        scene.restart()
169.    finally :
170.       sock.close()
171.
172.move()
```

**Figura 27.** Script Python Blender modificado para terreno infinito.
Fuente: Autor del proyecto.

En la Fig. 28, se puede ver un ejecutable de Blender.

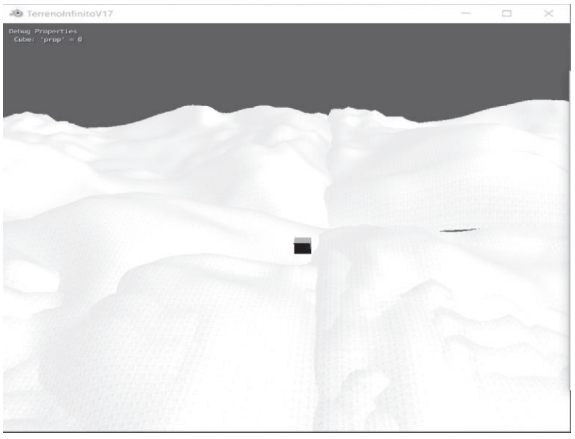

**Figura 28.** Ejecutable Blender.
Fuente: Autor del proyecto.

Para finalizar integramos Blender con el software desarrollado en Python, el cual se elegi la simulación deseada, en la integración de ambas herramientas es muy importante haber agregado los códigos Cliente y Servidor correctamente y de acuerdo a las necesidades.

Un ejemplo de una simulación es la de Movimiento Rectilíneo Uniforme, en la cual la partícula se desplaza con una velocidad y tiempo predeterminado. Ver Figura 29.

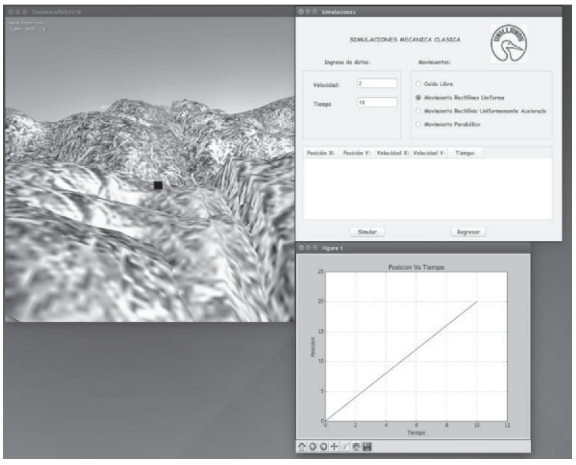

**Figura 29.** Integración Blender y software Python.
Fuente: Autor del proyecto.

## IV. CONCLUSIONES

Python y Blender son herramientas que en conjunto se puede obtener grandes resultados en la enseñanza de diferentes áreas como la ciencia, educación, procesos de manufactura, modelos matemáticos, demostración de los fenómenos físicos, entre otros campos de acción. Siendo así que la conexión de estas dos herramientas abre muchas posibilidades, ya sea para realizar cálculos complejos en otros lenguajes de programación o otra aplicación sin necesidad de sobrecargar el aplicativo de Blender, con lo cual este procedimiento o conexión permite un fundamento básico para el desarrollo de aplicaciones de simulación en 3D generando espacios interactivos para el aprendizaje en muchas áreas de estudio como se mencionó anteriormente. Además de esto se resalta que en el mundo del software libre hay mucho por explorar y generar aplicativos en conjunto para hacer competencia a cualquier software privativo.

Finalmente, los resultados de esta conexión permiten mejorar las simulaciones en general, siendo más interactivas y agradables para estudiantes si se tienen los modelos respectivos en tres dimensiones y aplicando tecnologías de libre acceso, por otra parte, como trabajo futuro es pensar en crear un software de ayuda para colegios de la región siendo de bajo costo.

## V. AGRADECIMIENTOS

Se agradece a la Dirección General de Investigaciones y al grupo de investigación Sistemas Dinámicos de la Facultad de Ciencias Básicas e Ingeniería de la Universidad de los Llanos por su apoyo en el desarrollo de este proyecto de investigación con código FCBI-13-2013.

## REFERENCIAS

Aguila, Z. (2013). PROJECT OF ANIMATED TRIDIMENSIONAL SOFTWARE AS ASSISTANT IN THE DEVELOPMENT OF SIMULTANEOUS ACTIVITY OF THE CEREBRAL HEMISPHERES. 17th International Congress on Project Management and Engineering.

Bacone, K. (2012). Blender Game Engine Begginers Guide. Birmingham: Pack Publishing Ltd.

Craig, D. (1996). Extensible Hierarchical Object-Oriented Logic Simulation with an Adaptable Graphical User Interface. Memorial Univ. of Newfoundland. Saint John, Canadá.

Ruiz, J. (2008). PYTHON 3D. Linux Magazine.Werther, P. V. (2013). METHOD FOR CARRYING OUT A MULTIMEDIA COMMUNICATION BASED ON A NETWORK PROTOCOL, PARTICULARLY TCP/IP AND/OR UDP .–

Daniel, J., Hernández, A., Fernando, A., López, J., Oswaldo, H., & Castro, P. (2016). DESARROLLO DE APLICACIONES EN PYTHON PARA EL APRENDIZAJE DE FÍSICA COMPUTACIONAL, 16, 7282.

Gharehchopogh, F. S., Amini, E., & Maleki, I. (2014). A New Approach for Inter Process Communication with Hybrid of Message Passing Mechanism and Event based Software Architecture. Indian Journal of Science and Technology, 7(6), 839847.

Ghate, P. V., & Pati, H. K. (2016). Collaborative distributed communication in hetero-geneous environments: A comprehensive survey. Journal of Network and Computer

Applications, 61, 120. https://doi.org/10.1016/j.jnca.2015.10.006

Martelli, A. (2008). Python: guía de referencia. Anaya Multimedia.

Morales, J. F. R. M. A. (2010). Experimentos numéricos en el aula sobre fenómenos difusivos: difusión anómala en sistemas físicos y biológicos, 56(1), 4150.

Rojas, J. F., Martínez, R., & Morales, M. A. (2014). Mecánica 3d: Python y el algoritmo de verlet. Revista Mexicana de Fisica E, 60(1), 5165.

Rojas, J. F., Morales, M. A., Rangel, A., & Torres, I. (2009). Física computacional: Una propuesta educativa. Revista Mexicana de Fisica E, 55(1), 97111.

Sarker, M. O. F., & Washington, S. (2015). Learning Python Network Programming. Packt Publishing Ltd.

Welch, B. B., & Jones, K. (2014). Socket Programming. Practical Programming in Tcl/Tk, 5(3), 237256.