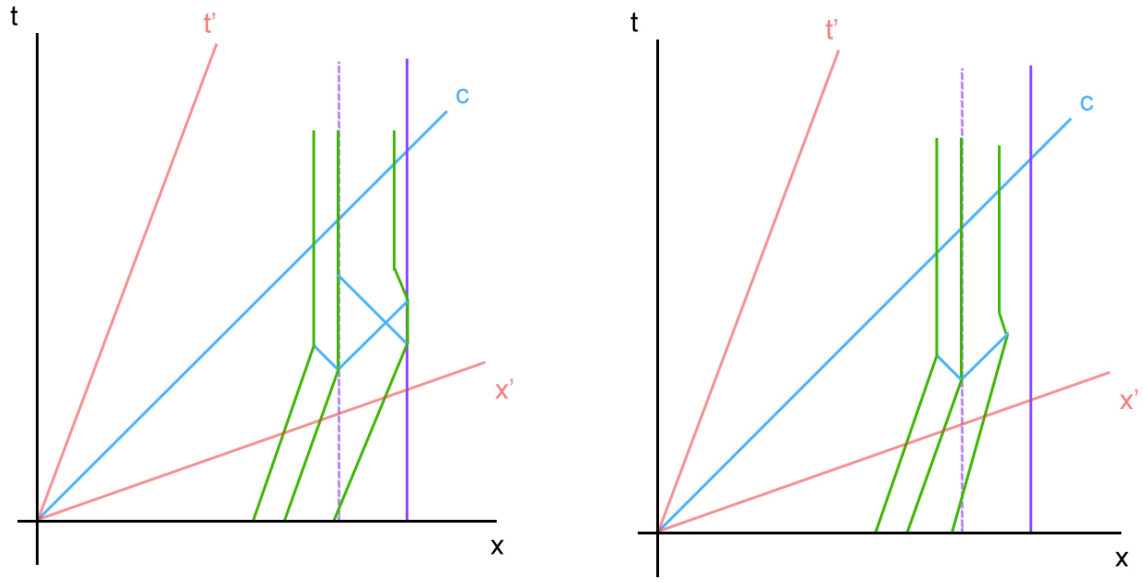
Preliminary Results

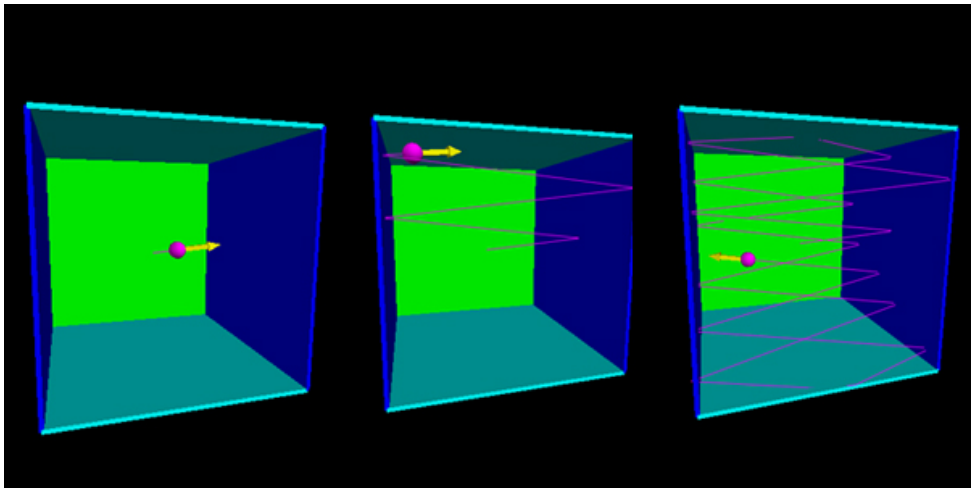
This document contains a collection of the results obtained thus far from the project. Explanations of diagrams are provided where necessary.

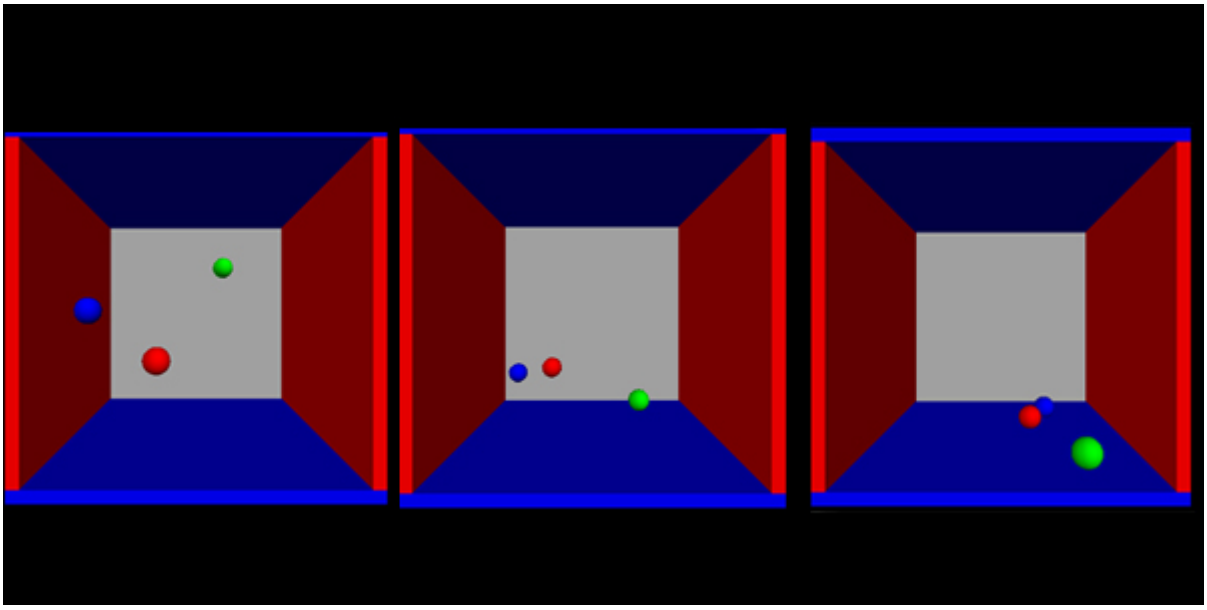
Rough Minkowski Diagrams



The above diagrams represent the speculative results of two possible outcomes of the lock and key paradox. The green lines represent the back, centre and front of the key, the purple lines represent the front and back of the lock, and the blue line represents the speed of light *c* (or the propagation of information throughout the moving key). The left diagram represents the scenario in which the key extends upon collision with the lock, allowing the front of the key to hit the back of the lock. The diagram on the right represents the scenario in which the propagation of information "outruns" the front of the key, preventing it from extending enough to make contact with the back of the lock.

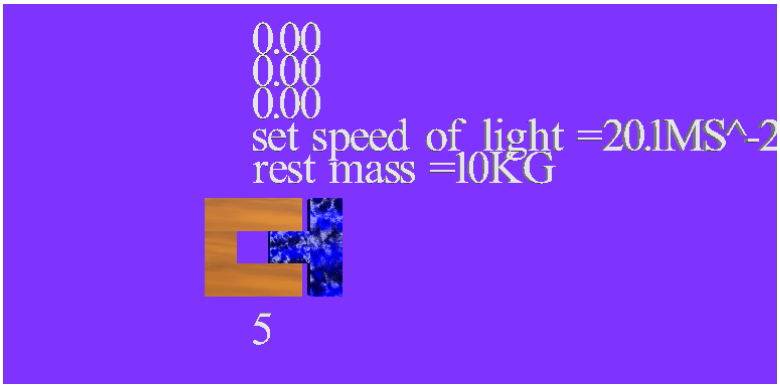
Basic Visual Python Simulations

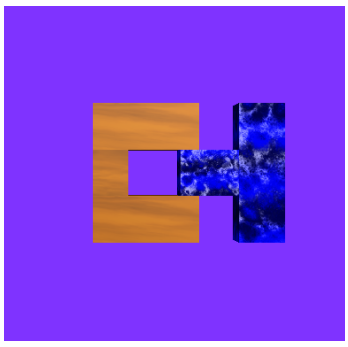


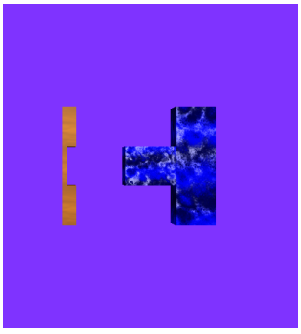


Above are screenshots of basic “particles in a box” models programmed in visual python. The first series of sreenshots represents an ideal gas particle elastically bouncing off the walls of the box, leaving a trail behind showing the path of motion. The second series of screenshots represents three balls starting in mid air undergoing the effects of very basic models of gravity and air resistance. The code used to produce these simulations are shown in the appendix below.

Below is a basic simulation of length contraction caused by relativistic speeds. Shown in the screenshots, the 'key' is in motion, causing the 'lock' to contract. For the sake of screenshots, both are shown directly beside each other throughout the simulation. The code used to produce this simulation is in the appendix below.







Appendix

First simulation

```
from visual import *
#define objects to be used in simulation in terms of postion, size and colour
ball = sphere(pos=(0,0,0), radius=0.5, color=color.magenta)
wallR = box(pos=(6,0,0), size=(0.2, 12, 12), color=color.blue)
wallL = box(pos=(-6,0,0), size=(0.2, 12, 12), color=color.blue)
wallBack = box(pos=(0,0,-6), size=(12, 12, 0.2), color=color.green)
wallTop = box(pos=(0,6,0), size=(12, 0.2, 12), color=color.cyan)
wallBottom = box(pos=(0,-6,0), size=(12, 0.2, 12), color=color.cyan)
#velocity vector of ball defined in x,y,z directions
ball.velocity = vector(25,3,5)
#change in time, or "refresh rate", of simulation defined, and initial time t defined
deltat = 0.01
t = 0
#vector arrow added to the ball, variable "vscale" used to scale it down to appropriate size
vscale = 0.1
varr = arrow(pos=ball.pos, axis=vscale*ball.velocity, color=color.yellow)
#auto scaling turned off, trail behind ball added
scene.autoscale = false
ball.trail = curve(color=ball.color)
#while loop defined within a time scale of 20
#as the walls defined above do not contain any physical information relative to the ball,
#if statements are used to define the wall boundaries of the ball's motion
while t < 20:
    if  ball.pos.x > wallR.pos.x:
        ball.velocity.x = -ball.velocity.x
    if  ball.pos.x < wallL.pos.x:
        ball.velocity.x = -ball.velocity.x
    if  ball.pos.y > wallTop.pos.y:
        ball.velocity.y = -ball.velocity.y
    if ball.pos.y < wallBottom.pos.y:
        ball.velocity.y = -ball.velocity.y
    if ball.pos.z < wallBack.pos.z:
        ball.velocity.z = -ball.velocity.z
    if ball.pos.z > 6:
        ball.velocity.z = -ball.velocity.z
        #in each frame of the program, the new position of the ball is defined
        #as the previous position added to the product of the velocity and the change in time
    ball.pos = ball.pos + ball.velocity*deltat
    varr.pos = ball.pos
    varr.axis = vscale*ball.velocity
    ball.trail.append(pos=ball.pos)
    t = t+deltat
    #the rate specifies that the while loop will not be executed more than 100 times per second
    rate(100)
```

Second simulation

```
from visual import *

print("""
Right button drag or Ctrl-drag to rotate "camera" to view scene.
```

```
Middle button or Alt-drag to drag up or down to zoom in or out.
  On a two-button mouse, middle is left + right.
""")
#Sets position and thickness of box
side = 4.0
thk = 0.3
#makes certain walls different dimentions so they dont overlap
s2 = 2*side - thk
s3 = 2*side + thk
#draws the walls of the box
wallR = box (pos=( side, 0, 0), size=(thk, s2, s3),  color = color.red)
wallL = box (pos=(-side, 0, 0), size=(thk, s2, s3),  color = color.red)
wallB = box (pos=(0, -side, 0), size=(s3, thk, s3),  color = color.blue)
wallT = box (pos=(0,  side, 0), size=(s3, thk, s3),  color = color.blue)
wallBK = box(pos=(0, 0, -side), size=(s2, s2, thk), color = (0.7,0.7,0.7))

#draws 3 balls of different colour
ball = sphere (color = color.green, radius = 0.4)
ball2 = sphere (color = color.red, radius = 0.4)
ball3 = sphere (color = color.blue, radius = 0.4)
#sets mass of the balls
ball2.mass = 1.0
ball.mass = 1.0
ball3.mass = 1.0
#sets initial force acting on balls
ball.p = vector (-1.15, -1.23, +1.27)
ball2.p = vector (-0.16, -0.22, +0.28)
ball3.p = vector (-0.54, -0.74, +0.76)

#Sets gravity, and chooses a coefficient for simple air resistance model
Gravity = -0.00981
air = 0.001
#error correction so the side of the ball hits the side of a wall and not the center of each
side = side - thk*0.5 - ball.radius

#sets time step interval and initial time
dt = 0.05
t=0.0
#while loop to run indefinetly
while True:
#sets the rate at which the program updates
  rate(500)
#steps through time
  t = t + dt

#Updates ball position based on the force and mass of the ball, (note its dt not dt^2 as dt is
counted once in the runthrough of the program)
  ball.pos = ball.pos + (ball.p/ball.mass)*dt
  ball2.pos = ball2.pos + (ball2.p/ball2.mass)*dt
  ball3.pos = ball3.pos + (ball3.p/ball3.mass)*dt
#Simple air resistance model based on the speed of the ball (not an accurate model, just a proof of
concept)
  ball.p = ball.p - air*ball.p
  ball2.p = ball2.p - air*ball2.p
  ball3.p = ball3.p - air*ball3.p
```

```
#if the ball is not on the ground let gravity act upon it
  if (side > ball.y > -side):
    ball.p.y = ball.p.y + Gravity
  if (side > ball2.y > -side):
    ball2.p.y = ball2.p.y + Gravity
  if (side > ball3.y > -side):
    ball3.p.y = ball3.p.y + Gravity

#if a ball hits a wall let it reverse the perpendicular velocity with some energy loss due to inelastic
collisions.
  if not (side > ball.x > -side):
    ball.p.x = -ball.p.x*0.98
  if not (side > ball.y > -side):
    ball.p.y = -ball.p.y*0.98
  if not (side > ball.z > -side):
    ball.p.z = -ball.p.z*0.98

  if not (side > ball2.x > -side):
    ball2.p.x = -ball2.p.x*0.98
  if not (side > ball2.y > -side):
    ball2.p.y = -ball2.p.y*0.98
  if not (side > ball2.z > -side):
    ball2.p.z = -ball2.p.z*0.98

  if not (side > ball3.x > -side):
    ball3.p.x = -ball3.p.x*0.98
  if not (side > ball3.y > -side):
    ball3.p.y = -ball3.p.y*0.98
  if not (side > ball3.z > -side):
    ball3.p.z = -ball3.p.z*0.98
```

Third Simulation

```
from visual import *
#sets the size and background colour of the window of the animation
scene.height = 600
scene.width = 1200
scene.background = (0.5,0.2,1)
scene.autoscale = False
time = 0

#draws the lock
mybox = box(pos = (0,0,0), length = 1, height = 1, width = 1, color = color.orange,
material=materials.wood)
mybox2 = box(pos = (1,-1,0), length = 3, height = 1, width = 1, color = color.orange,
material=materials.wood)
mybox3 = box(pos = (1,1,0), length = 3, height = 1, width = 1, color = color.orange,
material=materials.wood)

#draws the key
mybox4 = box(pos = (mybox.pos.x + 2.25,0,0), length = 1.5, height = 1, width = 1, color =
color.blue, material=materials.marble)
mybox5 = box(pos = (mybox.pos.x + 3.25,0,0), length = 1, height = 3, width = 1, color = color.blue,
material=materials.marble)
```

```
#example lock and key at origional size (off screen for debugging)
cmybox = box(pos = (0,10,0), length = 1, height = 1, width = 1)
cmybox2 = box(pos = (1,9,0), length = 3, height = 1, width = 1)
cmybox3 = box(pos = (1,11,0), length = 3, height = 1, width = 1)
cmybox4 = box(pos = (4,10,0), length = 1.5, height = 1, width = 1)
cmybox5 = box(pos = (5,10,0), length = 1, height = 3, width = 1)

#text to update the current values of mass, velocity and acceleration of the lock (mainly for
debugging and checking maths)
masstext = text(text='0.00', pos=(1,4,0))
acctext = text(text='0.00', pos=(1,6,0))
veltext = text(text='0.00', pos=(1,5,0))
ttext = text(text='10', pos = (1,-3,0))

#sets starting speed
r = vector(0.1,0,0)
#sets a speed of sound value
c=20.1
ctext = text(text='set speed of light =' +str(c)+ 'MS^-2', pos=(1,3,0))
#sets rest mass value
massor=10
mtext = text(text='rest mass =' + str(massor) + 'KG', pos=(1,2,0))
#sets a constant applied force
force=2
#sets a countdown timer before program runs
time = 10
while time >=1:
    rate(1)
    ttext.text = str(time)
    time = time -1

#runs program until speed of lock is just under the speed of light
while r.x < c-0.1:
#speed at which the while loop runs
    rate(10)
#length contraction
    mybox.length=1*sqrt(1-((r.x*r.x)/(c*c)))
#updates position of the lock
    mybox.pos = r+mybox.pos
#mass increasing
    mass=massor/sqrt(1-((r.x*r.x)/(c*c)))
#updats position and length contracts side of the lock, corrects for contraction around center
    mybox2.pos = ((mybox.pos.x+sqrt(1-((r.x*r.x)/(c*c)))),-1,0)
    mybox2.length=3*sqrt(1-((r.x*r.x)/(c*c)))
#updats position and length contracts side of the lock, corrects for contraction around center
    mybox3.pos = ((mybox.pos.x+sqrt(1-((r.x*r.x)/(c*c)))),1,0)
    mybox3.length=3*sqrt(1-((r.x*r.x)/(c*c)))

#calculates new increase in acceleration bassed on constant force and changing mass
    a=force/mass
#new speed, note acceleration can be added to velocity as the loop updates every unit of time.
    r.x=r.x + a

#updates screen postion
```

```
    scene.center = mybox.pos
#Updates dummy key position to easily show it compared to lock.
    mybox4.pos = ((mybox.pos.x + 2.25),0,0)
    mybox5.pos = ((mybox.pos.x + 3.25),0,0)

#below is to update text, not needed unless debugging at the moment

   #  masstext.text='current mass =' + str.format('{0:.2f}',mybox.pos.x) +'KG'
   # masstext.text='current mass =' + str.format('{0:.2f}',mass) +'KG'
   # acctext.text='current acceleration =' + str.format('{0:.2f}',a) +'MS^-2'
 #   veltext.text = 'current velocity =' + str.format('{0:.2f}',r.x) + 'MS^-1'
 #   veltext.pos=(mybox.pos.x,5,0)
 #   ctext.text ='set speed of light =' +str(c)+ 'MS^-2'
 #   ctext.pos=(mybox.pos.x,3,0)
```