Tutorial Vpython

1 Presentación de La Biblioteca *Visual Python*

Características generales

- "Programación 3D para simples mortales". "VPython facilita la creación de animaciones y visualizaciones 3D navegables, incluso para aquellos con poca experiencia en programación. Como está basado en Python, también tiene mucho que ofrecer a programadores e investigadores experimentados". `https://vpython.org/`
- Tiene una serie de elementos geométricos ya preparados: `https://glowscript.org/docs/VPythonDocs/primitives.html`
- Dispone de varias herramientas de interacción con el usuario: `https://glowscript.org/docs/VPythonDocs/controls.html`
- Permite la construcción de gráficos dinámicos de varios tipos: `www.glowscript.org/docs/VPythonDocs/graph.html`
- Permite la construcción de arreglos estáticos en 3D.
- Animaciones con movimientos preestablecidos.
- Simulaciones mediante discretización de ecuaciones diferenciales.

Uso de la Plataforma Glowscript

¿Para que sirve?

"GlowScript es un entorno potente y fácil de usar para crear animaciones 3D y publicarlas en la web. En Glowscript.org puedes escribir y ejecutar programas GlowScript directamente en tu navegador. guárdalos en la nube

de forma gratuita y compártelos fácilmente con otros".

¿Cómo usar?

- Acceda al sitio web de GlowScript: Glowscript.org
- Haga clic en Iniciar sesión en la esquina superior derecha.
- Utilice su cuenta de Google para iniciar sesión. De lo contrario, cree una cuenta de Google.
- Si el inicio de sesión funcionó, la información Iniciado sesión como "su inicio de sesión" (Cerrar sesión) debería aparecer en la esquina superior derecha de la pantalla.
- Haga clic en "su inicio de sesión" para acceder a su área de archivos.
- Haga clic en Agregar carpeta para crear una carpeta y organizar mejor sus archivos. Al nombrar la carpeta, desmarque la opción Público si desea que los archivos que contiene se mantengan privados.
- Haga clic en Crear nuevo programa para comenzar a escribir su programa en VPython.

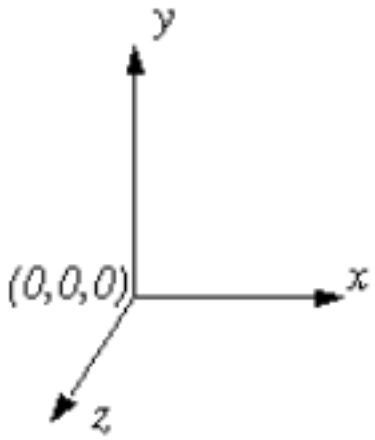
2 Geometrías y Operaciones Básicas de VPython

Creando un cubo y cambiando la vista de la escena generada.

- Después de crear un nuevo programa, escriba: box(). Luego haga clic en "*Run this program*" (Ejecutar este programa) para ejecutar el programa o presione **Ctrl + 1** en su teclado.
- Para rotar el ángulo de visión de la escena, mantenga presionado el botón derecho del mouse y arrástrelo.
- Para cambiar el zoom de la escena, utilice el botón de desplazamiento del ratón.
- Para mover la vista de la escena lateralmente, arrastre el mouse con Shift y el botón izquierdo presionado.

- Tenga en cuenta que todas estas operaciones no mueven ni rotan el objeto, sino sólo su ángulo de visión. Cambiar las propiedades del objeto

- La vista de la escena se describe mediante el sistema de coordenadas siguiente, donde el eje z apunta en dirección opuesta a la pantalla:



Las posiciones y desplazamientos vienen dados por la función vec.
Ejemplo: **v1 = vec(x, y, z)**

- Creemos un cubo, una esfera y una flecha conectando uno con el otro:

```
pos_cubo = vec(-2, -2, -2)
pos_bola = vec(2, 2, 2)
cubo = box (pos=pos_cubo,
            size=vec (1 ,2 ,3),
            color=color.green)
bola = sphere(pos=pos_bola,
              radius = 0.7,color=color.cyan )
flecha = arrow(pos=pos_cubo,
              axis=pos_bola-pos_cubo )
```

Si queremos cambiar alguna propiedad del objeto creado basta con usar: variable objeto.propiedad = valor.

- **Ejemplo 1:** cambiar el color de la flecha: flecha.color = color.amarillo

- **Ejemplo 2**: cambiar la posición del cubo: cube.pos = vector(-2.5,-1,-3)

- Si siempre quisimos vincular la flecha al cubo y la bola, deberíamos haber creado la flecha como: flecha = flecha(pos=cubo.pos, eje=bola.pos-cubo.pos)

3 Algunas Operaciones con Vectores

Creamos dos vectores:

```
vx = vec ( 1, 0 ,0)
vy = vec ( 0 , 1 , 0 )
vector_x = arrow ( axis=vx , color=color.green )
vector_y= arrow (axis=vy, color=color.cyan )
```

Calculamos el producto vectorial entre ellos:

```
vz = cross ( vx,vy)
vector_z = arrow ( axis=vz , color=color.red )
```

Ahora, hagamos una descomposición de un vector v1 en sus componentes paralelas y perpendiculares al vector v2:

```
v1= vec (-1.0 , 3.1 , 1.5 )
v2 = vec ( 3.0 , 3.0 , 2.0 )
v1pa = v1.proj ( v2 )
v1pe = v1 - v1pa
arrow ( axis = v1 , color = color. yellow , shaftwidth = 0.3
   )
arrow ( axis = v2 , color = color.blue , shaftwidth = 0.3  )
arrow ( axis = v1pa , color = color.white , shaftwidth = 0.3
   )
arrow ( axis = v1pe , color = color.orange , shaftwidth = 0.3
    )

a1 = vertex ( pos = vec ( 0 , 0 , 0 ) )
a2 = vertex ( pos = v1pa )
a3 = vertex ( pos = v1 )
a4 = vertex ( pos = v1pe )
quad ( vs = [ a1 , a2 , a3 , a4 ])
```

4 Simulaciones a partir de discretización de las ecuaciones diferenciales

Seguimos con nuestro sistema masa-resorte. Las variables dinámicas las vamos a calcular usando la 2 ley de Newton.

- $m\frac{\Delta v}{\Delta t} = F$
- $m\left(v_n - v_{n-1}\right) = F\Delta t$
- $\boxed{v_n = v_{n-1} + \frac{F\Delta t}{m}}$
- $\frac{\Delta x}{\Delta t} = v$
- $x_n - x_{n-1} = v_n\Delta t$
- $\boxed{x_n = x_{n-1} + v_n\Delta t}$

Necesitamos conocer la posición inicial y la velocidad inicial. Le vamos a agregar unos botones para que la simulación tenga Pausa, Continuar y Reiniciar. El código sería algo así:

```
s = "grafico de desplazamiento de sistema masa-resorte"
grafico = graph(title = s,
                xtitle = "tiempo[s]",
                ytitle = "Amplitud [u.a.]",
                fast = True,
                width = 800
)
curva = gcurve(color=color.blue,
               width = 4,
               markers = False,
               marker_color = color.orange,
               label = "curve"
)
mesa = box(pos = vec(0, 0, -0.15),
           size = vec(3, 2, 0.3),
           color = color.cyan
)
apoyo = box(pos = vec(1.35, 0, 0.25),
```

```
            size = vec(3, 2, 0.5),
            color = color.cyan
)

resorte.k = 30.0
bloque.b = 1.0

x0 = 0.8
v0 = 0.0

print("w0 =" + str(sqrt(resorte.k/bloque.masa)) + "rad/s")
print("gama =" + str(bloque.b/(2*bloque.masa)) + "rad/s")

dt = 0.01
t = 0.0
x = x0
v = v0

andando = 1

def Pausa(b):
    global andando
    andando = 0

def Continuar(b):
    global andando
    andando = 1

def Reiniciar(b):
    global andando
    andando = 2

button(text = "Pausa",
       pos = scene.title_anchor,
       bind = Pausa
)
button(text = "Continuar",
       pos = scene.title_anchor,
       bind = Continuar)

while True:
    sleep(t)
    #print(andando)
    if andando > 0:
        if andando == 2:
```

```
            t = 0.0
            x = x0
            v = v0
            curva.delete()
            andando = 1
        bloque.pos = vec(x, 0, 0.25)
        v += -(resorte.k*x + bloque.b*v)*dt/bloque.masa
        x += v*dt
        t = t + dt
        resorte.axis = bloque.pos - apoyo.pos
        curva.plot(t, x)
```

Apéndice

Objetos:

arrow (flecha)
box (prisma)
cylinder (cilindro)
helix (espiral)
sphere (esfera)
cone (cono)

Atributos:

pos (posición)
color (color)
axis (eje o dirección)
radius (radio)

Colores:

cyan (cian)
red (rojo)
yellow (amarillo)
orange (naranja)
black (negro)
magenta (magenta)
white (blanco que es el color por defecto)
green (verde)

ejemplo:

```
arrow(pos = vector(3,-2,5), axis(0,1,1), color =color.yellow)
box(pos = vector(0,0,1), size = vector(L,H,W), axis = (3,0,1)
    )
#donde L es el largo, H es la altura y W es la profundidad
```