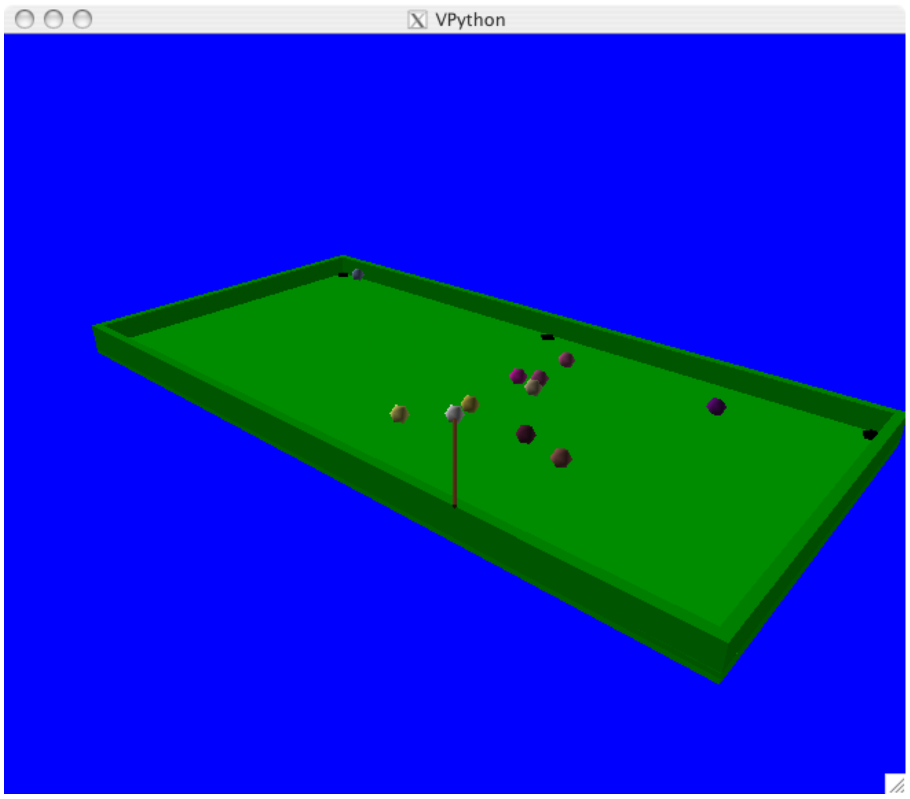
vPool Project

Copied from: https://www.cs.hmc.edu/twiki/bin/view/CS5/PeachyPool on 3/22/2107



Overview of the vPython project

For this project, you will be writing a fully interactive 3D "pool-like" game using vPython. We say "pool-like" because the game need not satisfy the official rules of pool and **the game may be substantially less complicated than a full pool game.** In fact, "interesting" interpretations and wide variations in game play are welcome!

This project need not be pool, but should be a 3d game using vpython, with

- at least four gameplay objects (of some design)
- at least two kinds of implemented collisions, e.g., point-to-line and point-to-point collisions
- some kind of user-control

- a game objective!
- a text file (`milestone.txt` and `final.txt` respectively) on how to play the game
- ***optional*** it never hurts to have a "grutor" mode that makes the game a bit easier to pay (and/or win)

Milestone requirements for the vPython project

For the milestone, you should submit a version for which the following pieces are implemented and work:

- A text document called `milestone.txt` with your name and the name of your partner (if any), the project that you've chosen, and a brief description of your plans for your game. One paragraph will suffice for the description. Also, include a description of the objective of the game - ***and how we should run it!***
- [Note for the final version - please include a file named `final.txt` in your final version that lets us know how to play and any special features or "gotchas" in the game.]
- In addition, your document should give instructions for playing the game as it currently operates, e.g. "Pressing the "s" key will do one thing, the mouse will allow the user to do another thing, etc."
- Implementation of enough of the functions, with docstrings, that the game does *something interesting* (even if it is far short of the complete implementation) in a file called `milestone.py` -- in fact, if your file is named something else, this is completely OK -- just note how we should run it in your `milestone.txt` file.
- Specifically, your program should include at least some "linear" collisions (a point object with a plane or a line) and at least some "spherical" collisions (a point object with another point object)
- Include any other files needed to test your game, as well... all of these should be zipped into `vpython_milestone.zip` and submitted.

Do be sure to zip up the folder containing your vPython project file (or files) and name it `vpython_milestone.zip` -- then submit that in the right spot in Hw#12!

Getting started with with vPython

Documentation on vPython is available at the vPython website. It should work pretty smoothly on Mac OS and Windows -- if you're installing onto your own machine or would like to run it on the CS Lab Macs, see some of the notes on the Lab 11 page.

Don't start from scratch!

Don't start this project with an empty Python file!!

Instead, start from your Lab 11 code - and then make changes to those files to implement your game!

The reason for this is that you will absolutely want to run your code with every change you make -- just to be sure it works. Don't write a large amount of code and then try to test it -- there will be too many opportunities for things to go wrong; in the end, you'll need to remove it all and bring back each small change and test as you go... !

Examples to use...

In addition to Lab 11, here are a few small example programs that show

- **wall_collide** a function to detect if a sphere (or point) collides with a wall (or any length/width box)
- **using keyboard to move the camera view** a good starting point for changing the view *programmatically*, e.g., as a character moves.
- **using mouse-drags to move/reorient an arrow** you can then print the vectors you create (hit `p`), but you may want to adapt the axis vector to become cueball velocity, for example.
- **orbital motion with vPython** along with an arrow showing the orbiting body's acceleration
- **how to implement ideal elastic collisions** hit spacebar to show this for 2 moving spheres - you may want to alter/expand this to more spheres...!

- **how to shoot snowballs in VPython** (hit space; the snowball will disappear when it gets far enough from the origin)
- **Using sounds from lab 3 in VPython...** This example (and zip file) will get you started incorporating sounds using the code from lab 3 (you'll have to convert things to .wav!)
- **here is the folder of examples that comes with vPython** - there are many to inspire ideas (and you can borrow the physics from them for your game...)

All of these programs should be runnable from VPython.

Requirements

This is, perhaps, the most open-ended of the final project options. Even so, there are a number of requirements -- as well as plenty of freedom to personalize your game!

Here are the requirements (and a few "non-requirements") for your game:

- When the game is started, it should have a 3-dimensional "table" along with a collection of "pool balls." Your game can be very different than pool, but should have some staging area and some "actors" or "agents" within it. Just to standardize, we'll use "pool balls" for whatever characters you choose in your game.
- There should be a "cue ball," that is, some object that the player can strike or control directly, and at least 3 additional objects with which the cue ball can interact.
- Three should be "walls," or some other obstacles, delimiting the game field through which game objects can not pass. They should support "linear collisions," that is collisions with an object defined by a point and an extended object (a line or plane). A ball that hits a wall should use the "angle of incidence equals angle of reflection" rule.
- There should be some pockets in the table where the balls can fall -- or some alternative notion of "goals or destination." Specifically, there should be some point-to-point interactions modeled and implemented, as well as the point-to-line interactions mentioned above.
- To summarize the interactions needed, your program should handle point collisions and bounces in some way. That is, the different objects in the game should interact visibly during the gameplay. This is where your game should support at least some "spherical collisions," that is,

collisions between two objects defined by points. They don't need to be spheres, but they're welcome to be.

- Specifically, balls/characters should not be able to pass through one another. It's nice to simulate the physics of ball-on-ball collisions using physical accuracy, but you may use other approximations or "wacky physics" for the spherical collisions. Just don't tell Profs. Gerbode and Esin that we used the words "wacky" and "physics" in the same sentence. Or, feel free to let them know that it was Prof. Kuenning who wrote this... .
- However, your game should allow for multiple balls/objects moving "simultaneously". For example, if ball `X` hits ball `Y` then ball `X` presumably continues moving (in some direction) and ball `Y` begins moving. Both of these balls should be moving for awhile. Moreover, ball `X` and ball `Y` may now hit other balls and these other balls may all need to move for some time as well. In other words, it is not acceptable for your program to always have at most one ball moving at a time.
- See the hint on how to compute collisions for several balls simultaneously below... .
- You should have a well-defined game objective with clearly specified win conditions. It can be a one-person game, a two-person game, or a game against the computer opponent. That's totally up to you!
- ***optional*** It never hurts to have a "grutor" mode that makes the game easier to play or win (or a "professor" mode in which you win immediately!)
- Your program should somehow alert the user when he/she has won the game!
- Your program must have an easy-to-use VPython interface (some combination of mouse and keyboard) that allows the user to hit the cue ball in different directions (optionally, with varying velocity). You *do not* need to have an actual "cue stick," but you're certainly welcome to do so. There are lots of other ways to handle the interface and this is entirely up to you.
- Your milestone and final project documentation should include a plain-text `milestone.txt` or `final.txt` file that describes the objective of your game and how to use the user interface. If you want, a short summary of the controls can also appear in the game itself... .

And, of course, feel free to add additional features as you wish... !

Handling friction...

- ***Warning*** Sometimes, vPool authors model friction by multiplying velocities by a constant less than 1 each time step. Note that the velocities never get to zero this way! ***It can be frustrating*** if it takes a very long time for objects to come to rest - before making another shot, for example. One way to avoid frustrating yourself (and the CS 60 graders!) is to check for slowly-moving objects and then simply set their velocities to 0. Once all velocities are zero, the next "move" in the game may be made.

Here are a few tips that may be helpful to you:

- First, remember that the vPython reference page is a self-contained resource for vPython.
- This vpython_events.py file is an example of one way to manage motion, keypresses, and mouse events
- The vPython `arrow` object can be used instead of drawing a pool cue to indicate the direction in which you will strike the cue ball. For example, the user might use the mouse or keyboard to rotate an arrow centered at the cue ball. Then another user input, such as hitting the space bar, could be used to cause the cue ball to move.
- See the vPython reference page for information on how to get keyboard and mouse input from the user... .
- Using a vector to keep track of the direction of movement of a ball allows for easy animation. Moreover, if the ball collides with another object then you need only update the vector appropriately.

Notes and hints

How do I handle all the collisions for lots of different objects?!

It may seem like you would need a zillion conditional statements to handle collisions for many objects, but you don't have to do all of them separately!

What many people do in this situation is to first create their objects:

```
ball1 = sphere(...stuff..)
ball2 = sphere(...stuff..)
ball3 = sphere(...stuff..)
ball4 = sphere(...stuff..)
...
```

and then make a list: `L = [ ball1, ball2, ball3, ball4, ... ]`

and then, to compute collisions, they run two loops through the list. Note that we only check each pair once here!

```
for i in range(len(L)):
  for j in range(i+1,len(L)): # note we start at i+1
     if  collide( L[i], L[j] ) == True:
        # do something appropriate...
```

That way you can write a single function (`collide`) to check if the two balls collide and then use it for all pairs....

What to Submit for the Milestone and Final Project

For the milestone, you should submit a version for which the following pieces are implemented and work:

- A text document called `milestone.txt` or, for the final project, `final.txt` with your name and the name of your partner (if any), the project that you've chosen, and a brief description of your plans for your game. One paragraph will suffice for the description. Also, include a description of the objective of the game - ***and how we should run it!***
- In addition, your `final.txt` document should give instructions for playing the game as it currently operates, e.g. "Pressing the "s" key will do one thing, the mouse will allow the user to do another thing, etc."
- [For the milestone]   Implementation of enough of the functions, with docstrings, that the game does *something interesting* (even if it is far short of the complete implementation) in a file called `milestone.py` -- in fact, if your file is named something else, this is completely OK -- just note how we should run it in your `milestone.txt`file.
- Specifically, your program should include at least some "linear" collisions (a point object with a plane or a line) and at least some "spherical" collisions (a point object with another point object)
- Include any other files needed to test your game, as well... all of these should be zipped into `vpython_milestone.zip` for the milestone or `vpython_final.zip` for the final version, and then submitted.
- [For the final]   you should submit a single `final.zip` file that includes the following:

- A text document called `final.txt` that provides a description of the game and its objectives and complete instructions for how to play the game.
- Your full game in a file called `final.py`.
- Be sure to include any other files needed to play your game, e.g., sounds, other python files that get imported, etc.