```
import viz
import vizact
import vizmat
import viztask
import vizinfo
import hand


# Various constants that are used in this file which you might want to tweak

GRAB_SELECT_POSSIBLE_ALPHA = .2 # Transparency to use for candidate selection bounding box
es
GRAB_SELECT_ALPHA = .4          # Transparency to use for objects that have been selected

WEIGHT_OF_CURRENT_VELOCITY = .5 # Used for computing velocity, not really used anywhere at
 the moment though

BB_EXTRA_SCALE = 1.0    # Extra scaling to apply to bounding box to make sure it is a bit
larger than the object inside it
SNAP_DISTANCE = 0.2     # Minimum distance to snap object to the end position
ANGLE_MIN_DIFF = 10;


class GrabHandList:
    """
    This is a wrapper class that takes in a hand list and processes each one with GrabHand
.
    This is the main object you should be using, or an inherited version. Do not call Grab
Hand directly.
    @param objlist:  a list of shapes you would like to control using the hands
    @param handlist: a list of hands, this would typically be acquired using a VUManager g
etHandList() method
    @param springs:  (default=True) tells the code to use physics springs instead of direc
t grab, springs enforce proper movements
    @param grabCenter: (default=False) should the grab always link the spring to the cente
r of the object, or wherever we do the grab (make sure the radius doesn't extend too far f
rom the object!)
    @param grabPosOnly: (default=False) when grabbing, normally we link the orientation to
 the hand. With this set to True, the object can rotate freely around a position only.
    @param highlight: (default=True) should we put a transparent box around the selection
    @param rotateFitBox: (default=False) try to rotate the bounding box to best fit the ob
ject, use the object rotation to help (currently not implemented!)
    """
    def __init__(self, objlist, handlist, *args, **kw):
        debug = False
        self.ListOfGrabHands = []
        # Traverse through each hand in the list and create a GrabHand class to handle eac
h one
        for eachhand in handlist:
            test_hand_obj = eachhand.getGesture() # Debugging to test if the supplied obje
ct is a valid hand object or not, if this fails you will get an exception
            if eachhand.configDebugPrimitives:
                debug = True
            newGrabHand = GrabHand (objlist, eachhand, *args, **kw)
            self.ListOfGrabHands.append(newGrabHand)
        # Show useful debugging information if we are in debug mode
        if debug:
            info = vizinfo.add("""
Hand debugging is enabled:
Yellow spheres show pinching detector sizes
```

```
Green sphere shows the selection region, linked to either the fist or finger tip
Red sphere shows punching solid, which is only active when the fist is enabled
""")
            info.drag(viz.ON)
            info.scale(0.75,0.75)



class PinchDetect:
    """
    This class handles detecting when a pinch gesture has been activated, testing the inde
x finger and thumb
    @param inputHand: The hand model to extract the finger tips from
    @param debug: Should we show debugging primitives or not
    @param pinchRadius: The radius of the finger tip primitives, these spheres must touch
to detect a pinch gesture
    """
    def __init__(self, inputHand, debug, pinchRadius):
        self.inputHand = inputHand
        self.pinchRadius = pinchRadius
        vizact.onupdate(viz.PRIORITY_PHYSICS+1,self.refresh)
        self.finger = inputHand.getFingerTipLinkable(hand.INDEX)
        self.thumb = inputHand.getFingerTipLinkable(hand.THUMB)
        self.pinched = False
        self.__lastgesture = hand.GESTURE_UNKNOWN
        self.debug = debug
        if debug:
            # Create yellow spheres to show the pinch regions
            import vizshape
            sphere = vizshape.addSphere(radius=pinchRadius)
            sphere.color(1,1,0)
            sphere.polyMode(viz.POLY_WIRE)
            viz.link (self.finger,sphere)
            sphere = vizshape.addSphere(radius=pinchRadius)
            sphere.color(1,1,0)
            sphere.polyMode(viz.POLY_WIRE)
            viz.link (self.thumb,sphere)

    def refresh(self):
        fpos = viz.Vector(self.finger.getPosition(viz.ABS_GLOBAL))
        tpos = viz.Vector(self.thumb.getPosition(viz.ABS_GLOBAL))
        diff = fpos - tpos
        if diff.length() < 2*self.pinchRadius:
            gesture = hand.GESTURE_PINCH_INDEX
            self.pinched = True
        else:
            gesture = hand.GESTURE_UNKNOWN
            self.pinched = False
        if self.__lastgesture is not gesture:
            #Create event and send it for this hand since there has been a change
            e = viz.Event()
            e.hand = self.inputHand
            e.gesture = gesture
            e.oldGesture = self.__lastgesture
            self.__lastgesture = gesture
            viz.sendEvent(hand.HAND_GESTURE_EVENT,e)
```

```
class GrabHand:
    """
    This class handles all the physics associated with a single hand grabbing an object. Y
ou should
    not use this class directly, instead use the GrabHandList class above which gets the l
ist of hands
    from vizuniverse automatically for you.
    """
    def __init__(self, grabNodes, inputHand, springs=True, grabCenter=False, grabPosOnly=F
alse, highlight=True, rotateFitBox=False):
        self.otherGrabbedNode = None
        # Save incoming settings
        self.springs = springs
        self.punching = True # Punching is always enabled, but only works if springs are e
nabled below
        self.punchRadius = 0.1 # For some reason, you cannot make this too small, or the p
hysics ball just jumps around too much
        self.grabCenter = grabCenter
        self.grabPosOnly = grabPosOnly
        self.highlight = highlight
        self.rotateFitBox = rotateFitBox

        # Some variables that we use the hand object to supply via vizuniverse create*Hand
*() methods
        if hasattr(inputHand, "configPinchEnable"):
            self.pinchGrab = inputHand.configPinchEnable
        else:
            print "hand.configPinchEnable not set, so using True by default"
            self.pinchGrab = True

        if hasattr(inputHand, "configFingerRadius"):
            self.pinchRadius = inputHand.configFingerRadius
        else:
            print "hand.configFingerRadius not set, so using 0.01 by default"
            self.pinchRadius = 0.01

        if hasattr(inputHand, "configTouchRadius"):
            self.grabRadius = inputHand.configTouchRadius
        else:
            print "hand.configTouchRadius is not set, so using 0.1 by default"
            self.grabRadius = 0.1

        if hasattr(inputHand, "configDebugPrimitives"):
            self.debug = inputHand.configDebugPrimitives
        else:
            print "hand.configDebugPrimitives is not set, so using False by default"
            self.debug = False

        # Scale all the radius values by the size of the hand in case it has been adjusted
        if hasattr(inputHand, "configHandSize"):
            self.handSize = inputHand.configHandSize
        else:
            self.handSize = 1.0 # Hands are usually 1.0 unless specially modified
        self.punchRadius *= self.handSize
        self.pinchRadius *= self.handSize
        self.grabRadius  *= self.handSize

        # Make a copy of the supplied input object list
        self.grabNodes = []
        for node in grabNodes:
```

```
            self.grabNodes.append(node)
        self.grabbedNode = None # No currently grabbed object

        # Store the supplied hand in this object
        self.handModel = inputHand

        # This is the box we show if the object is selected, hide it initially until neede
d
        self.selectNode = viz.add('box.wrl')
        self.selectNode.drawOrder(1000,bin=viz.BIN_TRANSPARENT) # Force this object to be
drawn later than anything else, because the depth sort is not always perfect
        self.selectNode.visible(0)

        # We create a bounding sphere to use for collisions with other objects
        import vizshape
        if self.debug:
            # Create a green sphere to show the collision region for finding matches with
other objects
            self.handSphere = vizshape.addSphere(radius=self.grabRadius)
            self.handSphere.color(0,1,0)
            self.handSphere.polyMode(viz.POLY_WIRE)
        else:
            self.handSphere = viz.addGroup()
        self.handSphere.collideSphere(radius=self.grabRadius)
        self.handSphere.disable(viz.PHYSICS) # Don't allow this object to move anything ar
ound

        # Decide whether we are doing pinch or fist-based grabbing
        if self.pinchGrab:
            # With pinches, attach the collider to the thumb tip, this is because the thum
b does not
            # move when in clicker mode, so you can be sure the bounding sphere will not c
hange. When
            # in motion capture mode, the finger and thumb will come together, so it doesn
't matter
            # whether we use the index or thumb.
            link = viz.link (self.handModel.getFingerTipLinkable(hand.THUMB), self.handSph
ere, enabled=False)
        else:
            # Use the whole hand, the user must form a fist to do a grab
            link = viz.link (self.handModel, self.handSphere, enabled=False)
        # Make sure the priority is right so the link is updated before we do the intersec
tion testing
        vizact.onupdate (viz.PRIORITY_PHYSICS+0, link.update)

        # Always create a punching physics object, but it may not be actually active later
 on
        if self.debug and self.punching:
            # Create a red sphere that we can use for punching things if needed
            self.punchSphere = vizshape.addSphere(radius=self.punchRadius)
            self.punchSphere.color(1,0,0)
            self.punchSphere.polyMode(viz.POLY_WIRE)
        else:
            self.punchSphere = viz.addGroup()
        self.punchSphere.collideSphere(radius=self.punchRadius, friction=0.0001, density=2
00)
        self.punchSphere.disable(viz.PHYSICS) # Don't allow this object to move anything a
round
        self.punchSphere.visible(0) # Only shown if the fist gesture is active
```

```
        # Add event handling for the hand, note the lambda notation below. It acts as a fi
lter so that
        # onGesture only gets called when the condition matches. See the vizact.onevent()
docs for more
        # info on why we use this.
        vizact.onevent(hand.HAND_GESTURE_EVENT, lambda e: (e.hand==self.handModel, e), sel
f.onFistGesture)
        if self.pinchGrab:
            PinchDetect(self.handModel, debug=self.debug, pinchRadius=self.pinchRadius)
            vizact.onevent(hand.HAND_GESTURE_EVENT, lambda e: (e.hand==self.handModel, e),
 self.onPinchGesture)

        # Recalculate the intersections and object motion every frame
        vizact.onupdate(viz.PRIORITY_PHYSICS+2, self.update)

        # Need to compute velocity so we can throw objects, possibly this code is not even
 used?
        self.lastPos = None
        self.localVelocity = vizmat.Vector([0,0,0])
        vizact.onupdate(viz.PRIORITY_PHYSICS-2,  self.computeVelocity)


    def onFistGesture(self, e):
        if e.gesture == hand.GESTURE_FIST:
            if self.debug: print "Gesture Fist Start"
            self.grab(hand.GESTURE_FIST)
        elif e.oldGesture == hand.GESTURE_FIST:
            if self.debug: print "Gesture Fist Stop"
            self.release(hand.GESTURE_FIST)

    def onPinchGesture(self, e):
        if e.gesture == hand.GESTURE_PINCH_INDEX:
            if self.debug: print "Gesture Pinch Start"
            self.grab(hand.GESTURE_PINCH_INDEX)
        elif e.oldGesture == hand.GESTURE_PINCH_INDEX:
            if self.debug: print "Gesture Pinch Stop"
            self.release(hand.GESTURE_FIST)

    def update(self):
        if self.grabbedNode:
            # If something is already grabbed, then re-position a transparent selection bo
x
            self.selectNode.alpha(GRAB_SELECT_ALPHA)
            self.positionSelectionBox(self.grabbedNode)
        else:
            # We have nothing grabbed, so lets see if we have an intersection
            grabItem = self.handIntersect()
            if grabItem:
                # If we found something, put a transparent selection box to indicate it is
 a candidate
                if self.highlight:
                    self.selectNode.visible(viz.ON)
                self.selectNode.alpha(GRAB_SELECT_POSSIBLE_ALPHA)
                self.positionSelectionBox(grabItem)
            else:
                # We did not find anything, so hide the transparent selection box for now
                self.selectNode.visible(viz.OFF)


    def positionSelectionBox(self, grabItem):
```

```
        bb = grabItem.getBoundingBox(viz.ABS_GLOBAL)
        self.selectNode.setScale([bb.width * BB_EXTRA_SCALE, bb.height * BB_EXTRA_SCALE, b
b.depth * BB_EXTRA_SCALE])
        self.selectNode.setPosition(bb.center)


    def handIntersect(self):
        list = viz.phys.intersectNode(self.handSphere)
        if len(list) > 0:
            selectedItemDistance = 999999
            selectedItem = None
            handPos = self.handSphere.getPosition(viz.ABS_GLOBAL)
            for item in list:
                if ((item in self.grabNodes) and (item != self.otherGrabbedNode)):
                    d = vizmat.Distance(handPos, item.getPosition(viz.ABS_GLOBAL))
                    if selectedItemDistance > d:
                        selectedItem = item
                        selectedItemDistance = d
            self.onIntersect(selectedItem)
            return selectedItem

        self.onIntersect(None)
        return None

    # Methods that can be overridden if you want to do things after an object is grabbed,
currently we do nothing
    def postGrabAction(self):
        #viz.disable(viz.DYNAMICS)
        pass
    def postReleaseAction(self):
        pass

    def onIntersect(self, selectedItem):
        pass
    def grab(self, gesture):
        # Are we already grabbing? (either pinch or fist already)
        if not self.grabbedNode:
            # We do not have anything in our hands
            self.grabbedNode = self.handIntersect()

            # Test to see if we should implement punching
            if self.punching and not self.grabbedNode and gesture is hand.GESTURE_FIST and
 self.springs:
                # Punching mode is on, gesture was a fist, and no object found, and spring
s are on, so lets activate our punching object
                self.punchSphere.enable(viz.PHYSICS)
                self.punchSphere.visible(1)
                self.punchSphere.setPosition (self.handSphere.getPosition(viz.ABS_GLOBAL),
 viz.ABS_GLOBAL)
                self.grabbedNode = self.punchSphere

            # Decide if we are grabbing with a pinch, or if we are doing a large scale fis
t grab
            if gesture is hand.GESTURE_FIST:
                target = self.handModel # Fists use the main hand, so when the thumb moves
 in on a fist event the object doesn't rotate
            else:
                target = self.handSphere # Fine pinch grabs, where the thumb rotation is u
sed
```

```
            # If we found a node to grab then lets go ahead and do it
            if self.grabbedNode:
                if self.springs:
                    self.startSpring(self.grabbedNode, target)
                else:
                    self.startGrab(self.grabbedNode, target)
                self.postGrabAction()
        else:
            # Object already being grabbed
            if self.debug: print "Last grab event ignored since object already grabbed"

    def release(self, gesture):
        if self.grabbedNode == self.punchSphere:
            self.punchSphere.disable(viz.PHYSICS)
            self.punchSphere.visible(0)
        if self.grabbedNode:
            # We have an object, so lets release it
            if self.springs:
                self.endSpring()
            else:
                self.endGrab()
            self.postReleaseAction()
            self.grabbedNode = None


    # Here is an implementation that uses pure viz.grab, it does not implement physics pro
perly, and you can push a can under the conveyor belt
    def startGrab(self, obj, target):
        #from RearSpar_V12 import OpenPart
        #if target in grabNodes:
        self.grablink = viz.grab (target, obj, absolute=True)
            #viz.enable(viz.PHYSICS)
            #self.grabbedNode.enable(viz.PHYSICS)
    def endGrab(self):
        distance = vizmat.Distance(self.grabbedNode.end_matrix.getPosition(), self.grablin
k.getDst().getPosition())
        self.grablink.remove()
        #roisin mcconnell 12th march put distance above end link
        angleDifference= vizmat.Distance(self.grabbedNode.end_matrix.getPosition(), self.g
rablink.getDst().getPosition())
        #TODO angle diference compare current
        #use quaternions instead of euler maybe
        #angleDifference = self.grabbedNode.end_matrix.getEuler() self.grablink.getDst().g
etEuler())
        angleDifference = 0
        if (distance<SNAP_DISTANCE):
            self.grabbedNode.setMatrix(self.grabbedNode.end_matrix)
            self.grabbedNode.disable(viz.PHYSICS)
        self.grabbedNode.enable(viz.PHYSICS)



    # This is a proper implementation of physics using springs, this is the preferred way
to do these things at the moment
    def startSpring(self, obj, target):
        # We need to setup a spring to implement the grab operation. Also, we need to sche
dule a task to
        # constantly update the spring dest position, you need to do this each frame as th
e hand moves around
        # For the constants, they work as follows:
```

```
        # linearKd - force applied by spring to the object, higher is more force
        # linearKs - dampening of the spring, higher values slow down the time it takes fo
r the object to react to the spring
        # angularKd - unsure, not documented
        # angularKs - unsure, not documented
        if self.grabPosOnly:
            self.spring = obj.addSpring(viz.LINK_POS, linearKd=70, linearKs=1000, angularK
d=.3, angularKs=3)
        else:
            self.spring = obj.addSpring(viz.LINK_ALL, linearKd=70, linearKs=1000, angularK
d=.3, angularKs=3)

        # Create a target object that we can map our desired position and orientation on t
o
        self.targetobj = viz.addGroup()
        self.targetobj.setMatrix(obj.getMatrix(viz.ABS_GLOBAL), viz.ABS_GLOBAL)
        self.springGrab = viz.grab (target, self.targetobj)
        self.springUpdate = viztask.schedule (self.repositionSpringTask(obj, target, self.
targetobj))

        # We use spring offsets if we want the grab to occur where the hand is currently,
not at the center of the object
        if self.grabCenter is False:
            # These lines set up springs. Both local and global seem to work now. Global g
rabs seem to explode sometimes if you push hard enough!
            #self.spring.setOffset([1,0,0], viz.REL_LOCAL) # Simple test of local grabbing
            self.spring.setOffset(target.getPosition(viz.ABS_GLOBAL), viz.ABS_GLOBAL) # At
tach spring at the location the hand is at


    def endSpring(self):
        self.spring.remove()
        self.springUpdate.kill()

    def repositionSpringTask(self, obj, destpos, destori):
        # Update the position of where the spring should connect to
        while True:
            self.spring.setPosition (destpos.getPosition(viz.ABS_GLOBAL))
            self.spring.setQuat (destori.getQuat(viz.ABS_GLOBAL))
            yield None

    # Recompute the velocity, which is needed for physics interactions for some reason? No
t sure if this even gets used.
    def computeVelocity(self):
        currentPos = vizmat.Vector(self.handSphere.getPosition(viz.ABS_GLOBAL))
        if self.lastPos:
            speed = vizmat.Distance(currentPos, self.lastPos) / viz.getFrameElapsed()
            currentVel = vizmat.Vector( vizmat.VectorToPoint(self.lastPos, currentPos), no
rmalized=True )
            currentVel *= speed
            currentVel *= WEIGHT_OF_CURRENT_VELOCITY

            self.localVelocity *= (1.0-WEIGHT_OF_CURRENT_VELOCITY)
            self.localVelocity = vizmat.Vector( currentVel + self.localVelocity )

        self.lastPos = currentPos


if __name__ == "__main__":
    ks = hand.KeySensor('',gestureDown=hand.GESTURE_FIST,gestureUp=hand.GESTURE_FLAT_HAND)
```

```
# Reversing the argument order causes problems
    ahand = hand.HandModel(ks)
    ahand.configFingerRadius = 0.01
    ahand.configTouchRadius = 0.1
    ahand.configPinchEnable = True
    ahand.configDebugPrimitives = True


    GrabHandList(objlist=[viz.addGroup()], handlist=[ahand])
    viz.go()
    viz.MainView.setPosition(-0.5,0.5,0.5)
    viz.MainView.lookat(0,0,0)
```