Electronic Supplementary Material (ESI) for Energy Advances.

Supporting Information

Total Energies of Electrons

Table S1 - total energies of the simulation for electrons, by slab and by dopant.

Dopant	*OH_2 Slab (eV)	*OH Slab (eV)	*O Slab (eV)	*OOH Slab (eV)
Undoped	2.0804	1.7528	-0.1106	1.5475
Al	2.3136	1.9174	-0.0797	1.8025
Ni	2.1156	0.5982	-0.6691	0.8126
Ti	2.0820	1.8885	-0.7439	0.9158

Cumulative Probability With the Same Kinetic Energy for All Intermediates

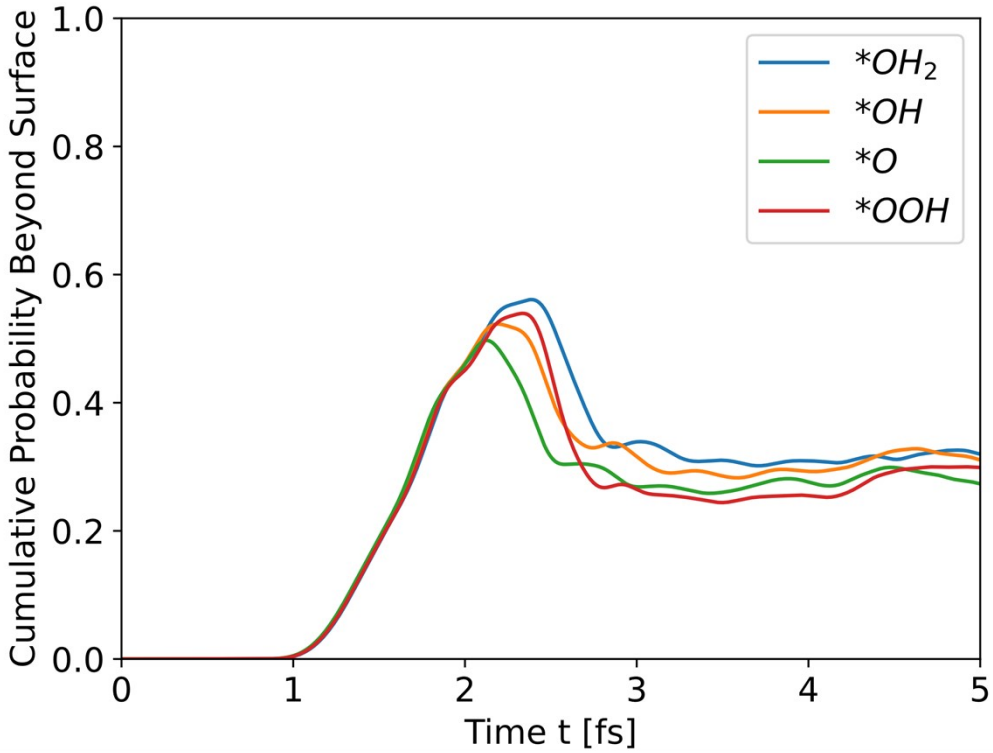

Figure S1 - cumulative probability beyond the surface for all intermediates with the same starting kinetic energy.

Cumulative Probability Beyond the Surface – Other Dopants

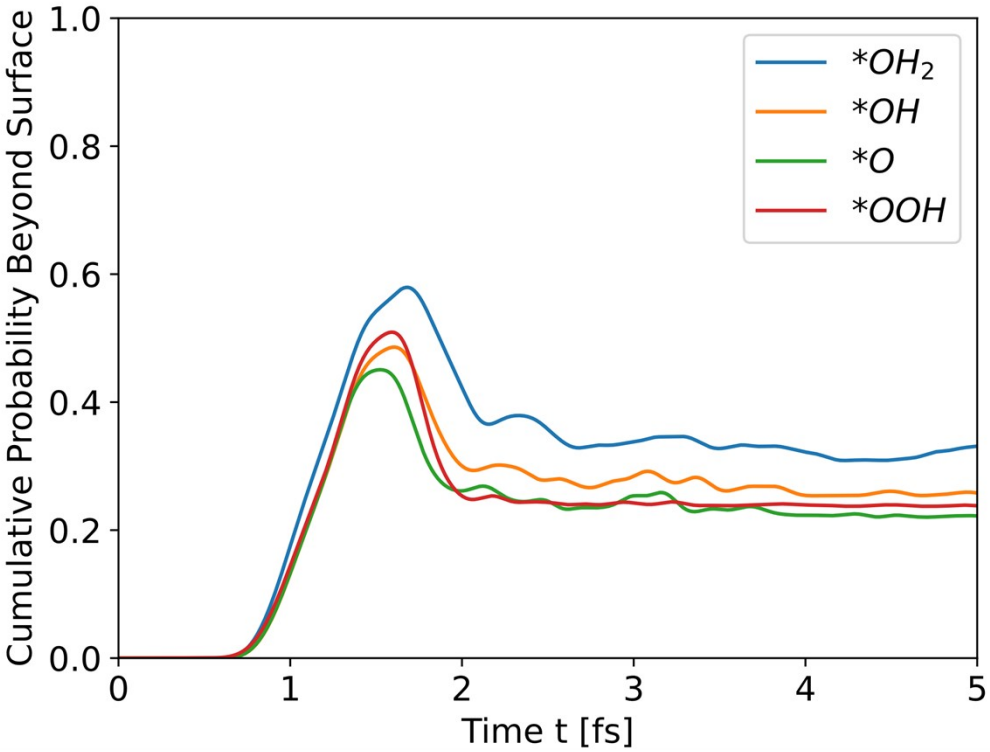

Figure S2 - cumulative probability beyond the surface, Al doped hematite. Hole propagation.

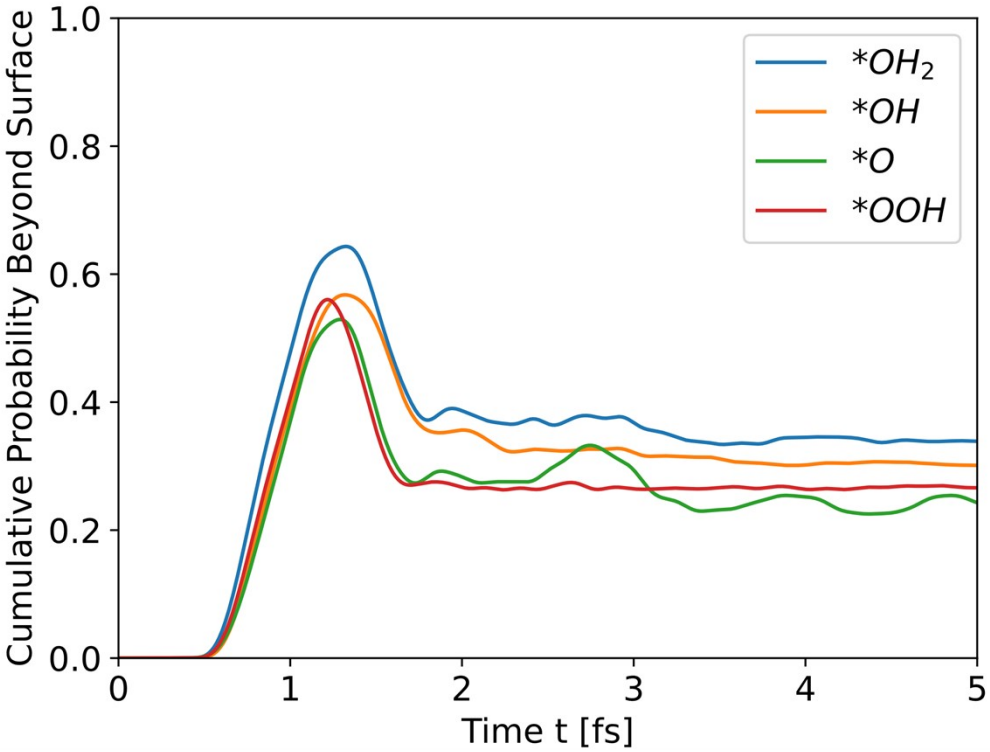

Figure S3 - cumulative probability beyond the surface, Ti doped hematite. Hole propagation.

INCAR File for LOCPOT Generation

```
SYSTEM= feouj0       # title of the task


# Dimensions of arrays

EMIN =  -20        #minimum energy for evaluation of DOS

EMAX =  20           #maximum energy for evaluation of
DOS

NEDOS=  20000         #number of grid points in DOS



# Start parameter for this run:


ISTART = 1           # job   : 0-new  1-cont  2-samecut #
changed from 1

ICHARG = 1          # charge: 1-file 2-atom 10-const

#INIWAV = 1          # electr: 0-lowe 1-rand  2-diag

PREC = Accurate      # medium, high low

ALGO = Very_Fast          #Normal,VeryFast...#specify the
electronic minimisation algorithm

#TIME= 0.4           #controls the trial time step for
IALGO=5X, for the initial (steepest descent) phase of
IALGO=4X


# DOS related values:

ISMEAR = -5          #-5,-4,-3,-2,0,N #0-Gaussian
smearing

                     #determines how the partial
occupancies f(n,k)are set for each orbital

SIGMA = 0          #broadening in eV -4-tet -1-fermi 0-
gaus

EDIFF=1E-4           #global break condition for the
electronic SC-loop

NELM=1500             #maximum number of electronic SC
(selfconsistency) steps

NELMIN = 6
```

```
LWAVE=.TRUE.          #These tags determine whether the
orbitals (file WAVECAR),

LCHARG=.TRUE.         #the charge densities (file CHGCAR
and CHG) are written

LVTOT=.TRUE.          # Write LOCPOT


# Spin and symmetry parameters

MAGMOM= 19*0 6*4.2 6*-4.2 4*0  #Specifies the initial
magnetic moment for each atom (order of POSCAR)

ISPIN=2               #1=non/2= spin polarized calculations
are performed

ISYM=0                #-1,0,1,2,3 #switch symmetry on
(ISYM=1, 2 or 3) or off (ISYM=-1 or 0)

LMAXMIX=4             #controls up to which l quantum
number the onsite PAW charge densities are passed through
the charge density mixer and written to the CHGCAR file


# Ionic relaxation

EDIFFG = -3.0E-2    #Def=EDIFF*10 defines the break
condition for the ionic relaxation loop

LORBIT=10             #11=>RWIGS in INCAR is not read.
DOSCAR and lm decomposed PROCAR file

NSW = 2000              #maximum number of ionic steps

IBRION = 2            #=-1 for NSW=0 or NSW=1 #determines
how the ions are updated and moved

#IOPT = 7

POTIM = 0.25          #For IBRION=1,2 or 3, POTIM serves as
a scaling constant for the forces

ISIF=0                #0-6 =0 if IBRION=0(MD)else =2
#controls whether the stress tensor is calculated

#ADDGRID = .TRUE.


LREAL=.False.         #projection done in:
FALSE=reciprocal/TRUE=real space

#LREAL = Auto          # Speed things up

#NSIM=1                #if spacified the RMM-DIIS
algorithm(IALGO=48)works in a blocked mode
```

```
NPAR=4              #~sqrt(number of cores) or number of
cores per compute node

KPAR=4              #KPAR is the number of k-points that
are to be treated in parallel

#NBAND              #Def: NELECT/2 + NIONS/2 (non-
spinpolarized)

                    #     0.6*NELECT + NMAG (spin-
polarized)

                    #determines the actual number of
bands in the calculation


ENCUT = 700 eV      #Cut-off energy for plane wave basis
set in eV. largest ENMAX from POTCAR-fil


#On site Coulomb interaction: L(S)DA+U

LDAU=.TRUE.         #TRUE-switches on the L(S)DA+U

LDAUTYPE= 2         #specifies which type of L(S)DA+U
approach will be used

LDAUPRINT= 0        #0,1,2. controls what to write in
OUTCAR

LDAUL= -1 2 -1 -1       #-1=no on-site terms added, 1= p,
2= d, 3= f

LDAUU= 0.0 4.6 0.0      #specifies the effective on-site
Coulomb interaction parameters

LDAUJ= 0.0 0.3 0.0     #specifies the effective on-site
Exchange interaction parameters
```

Wave Propagation Code

```
# Read LOCPOT

import numpy as np
import re
import scipy.signal as sig
import scipy.interpolate as sint

def extendPotential(potential, zaxis, numpoints):
    firstPoint = np.repeat(potential[0],numpoints)
    lastPoint = np.repeat(potential[-1],numpoints)
    dz = zaxis[1]-zaxis[0]
    newPot = np.concatenate((firstPoint,potential,lastPoint))
    zAdd = np.linspace(dz,2*numpoints*dz,2*numpoints)
    newZ = np.concatenate((zaxis,zaxis[-1]+zAdd))
    return newPot, newZ

def extendInternalPotential(potential,zaxis,numtimes):

    # Take the repeating unit of the potential and insert
multiple times

    # Find peaks of minus potential (find minimas)
    inds, props = sig.find_peaks(-potential,width=10)

    # The repeating potential
    exPot = potential[inds[1]:inds[-2]]
    dz = zaxis[1]-zaxis[0]
    newPot =
np.concatenate((potential[:inds[1]],np.tile(exPot,numtimes),poten
tial[inds[1]:]))
    newZ = np.arange(0,len(newPot)*dz-1e-9,dz)
    return newPot, newZ

def addVoltage(potential, zaxis, voltage):
    EField = voltage / (zaxis[-1]-zaxis[0])
    newPot = potential - EField * (zaxis - zaxis[0])
    return newPot

def ReadLOCPOT(FilePath, internalDups, vacuumEx, voltage):

    # Open file and read contents
    fid = open(FilePath,'r')
    fconts = fid.read()

    # Define regex to find dimensions
    findtext = ('\s* \d+ \s* \d+ \s* \d+')

    # Find all occurances of three integers in a line
    x = re.findall(findtext,fconts)

    # Take the last occurance and split into components
    xdim, ydim, zdim = x[-1].split()
    dimen = int(zdim), int(xdim), int(ydim)
    zpoints = dimen[0]

    # Now find the line where the potential begins
    newtext = '(.*)' + xdim + '(.*)' + ydim + '(.*)' + zdim +
'(.*)'
```

```
    # Delete file contents (to save memory)
    fconts = None

    # Rewind to beginning of file
    fid.seek(0)

    # Read entire file into a list by line
    linelist = fid.readlines()
    fid.close()

    # Get length of z vector
    # IMPORTANT: assumes cubic, tegragonal,
    # or hexagonal lattice
    zlength = float(linelist[4].split()[-1])

    # Create k-vector
    # Divide into dz - entire length by number of points
    dz = zlength / zpoints
    zaxis = (np.arange(zpoints)-1)*zlength/zpoints


    # Look for beginning of potential
    LineToStart = 0
    for i in linelist:
        LineToStart = LineToStart + 1
        if re.match(newtext,i):
            break

    # Initialize potential variable
    potential = np.zeros(np.prod(dimen))

    # How many lines to read (assuming 5 points
    # per line with overflow)
    LinesToRead = int(np.ceil(np.prod(dimen)/5))

    # Read potential into array
    k = 0
    for i in linelist[LineToStart:LineToStart+LinesToRead]:
        LineRead = i.split()
        for j in LineRead:
            potential[k] = float(j)
            k = k + 1

    # Reshape - z-axis FIRST (first index is sheets)
    potential = np.reshape(potential,dimen)

    # Average over xy axis
    avpot = np.sum(potential,axis=(1,2))/np.prod(dimen[1:])
    if (len(avpot) % 2 == 0):
        avpot = np.append(avpot,avpot[-1])
        zaxis = np.append(zaxis,zaxis[-1]+dz)

    """
    gridSize = int(np.round(len(zaxis)*np.round(dz/0.001+5,-1),-
3))
    intPot = sint.interp1d(zaxis,avpot)
    zaxis = np.linspace(zaxis[0],zaxis[-1],gridSize)
    avpot = intPot(zaxis)
    """
    # How many times to replicate the bulk
```

```
    avpot, zaxis =
extendInternalPotential(avpot,zaxis,internalDups)

    # How many points to add to each side of the potential
    avpot, zaxis = extendPotential(avpot,zaxis,vacuumEx)

    # Increase number of points
    zpoints = len(zaxis)

    EField = voltage / zaxis[-1]
    avpot = avpot - EField * zaxis
    # Return correct k-axis
    kaxis = np.fft.fftfreq(zpoints,dz)*2*np.pi

    return avpot, zaxis, kaxis


# All functions for ChargeProp

import numpy as np
from sympy.solvers import solve
from sympy import Symbol
from scipy.special import erf

# h-bar in eV * fs
hbarev = 6.58211951e-16 * 1e15

# h-bar in kg A^2 / fs
hbar = 1.0545718e-34 * 1e-15 * 1e20

# h-bar in J s
# Gives energy in J, k in 1/m
hbark = 1.0545718e-34

# electron mass in kg
me = 9.10938356e-31

# electron charge in C
q = 1.602176634e-19

def RunOneTimestep(wavefun, timestep, V, K):
    TempWavefun = np.exp(-1j*V*timestep/(2*hbarev))*wavefun
    TempWavefun = np.fft.fft(TempWavefun)
    TempWavefun = np.exp(-
1j*hbar*timestep*K**2/(2*me))*TempWavefun
    TempWavefun = np.fft.ifft(TempWavefun)
    TempWavefun = np.exp(-1j*V*timestep/(2*hbarev))*TempWavefun
    return TempWavefun

def calculateProbabilityCurrent(wavefun, k):
    dwavefun = deriveOne(wavefun, k)
    cwavefun = np.conj(wavefun)
    cdwavefun = np.conj(dwavefun)
    j = hbar/(2*me*1j) * (cwavefun*dwavefun - wavefun*cdwavefun)
    return j

def deriveOne(wavefun, k):
    wavefun = np.fft.ifft(1j*k*np.fft.fft(wavefun))
    return wavefun

def potentialExpectation(wavefun, potential, dx):
```

```
    norm = np.sum(wavefun*np.conj(wavefun)*dx)**0.5
    wavefun = wavefun / norm
    Vexp = np.sum(wavefun*np.conj(wavefun)*potential*dx)
    return Vexp

def initialMomentum(totalEnergy, wavefun, potential, dx, sigma,
L, x0):
    Vexp = np.real(potentialExpectation(wavefun, potential, dx))
    kineticEnergy = totalEnergy - Vexp
    A = np.sum(wavefun*np.conj(wavefun)*dx)
    k0 = Symbol('k0')
    k0 = solve(zFromGauss(k0,L,x0,sigma,kineticEnergy,A),k0)
    return k0

def zFromGauss(k0, L, x0, dx, kineticEnergy, A):
    x0 = x0 * 1e-10
    L = L * 1e-10
    dx = dx * 1e-10
    kineticEnergy = kineticEnergy
    sp = np.sqrt(np.pi)
    Lpart = (L-x0)/dx
    xpart = x0/dx
    erfpart = erf(Lpart) - erf(-xpart)
    expL = np.exp(-Lpart**2)
    expx0 = np.exp(-xpart**2)
    expLL = Lpart*expL
    expxx0 = xpart*expx0
    return -hbark**2/(2*me*A) * (1/(4*dx) * (sp*erfpart - 2*expLL
- 2*expxx0) - 1j*k0*(expx0 - expL) - sp*dx/2*(k0**2 +
1/dx**2)*erfpart) - kineticEnergy
    #return
(hbark**2/(2*me))*(A**2)*(1/(4*dx**2))*((1+2*dx**2*k0**2)*sp*dx*(
erf(x0/dx)+erf((L-x0)/dx)) + 2*np.exp(-
(x0/dx)**2)*(x0+2*1j*k0*dx**2) + 2*np.exp(-((L-x0)/dx)**2)*(L-x0-
2*1j*k0*dx**2))

def cumulativeProb(wavefun, x, xcutoff, dx):
    xAxis = x >= xcutoff
    cutwavefun = wavefun[xAxis]
    cumProb = np.sum(cutwavefun*np.conj(cutwavefun)*dx)
    return cumProb

def closestPeak(target, peakList):
    peakList = np.array(peakList)
    peakDiffs = np.abs(peakList - target)
    return int(peakList[peakDiffs == min(peakDiffs)][0])



import tkinter as tk
import numpy as np
from tkinter import filedialog
from ReadLOCPOT import ReadLOCPOT
import matplotlib.pyplot as plt
import matplotlib.animation as anim
from CPFunctions import *
import scipy.signal as sig

root = tk.Tk()
root.withdraw()
```

```
n = 5000
timestep = 0.001 # in fs
flt = 0
cumprob = np.zeros((n,4),dtype=np.cdouble)

#for fl in ['OHH', 'OH', 'O', 'OOH']:
for fl in 4*['O']:
    #FilePath = filedialog.askopenfilename(title="Choose LOCPOT
File")
    #FilePath = 'D:\OneDrive - Technion\Charge Transfer\LOCPOT-'
+ fl
    FilePath = 'G:\LOCPOT-' + fl
    Potential, ZAxis, KAxis = ReadLOCPOT(FilePath,5,5000,0)
    #Potential, ZAxis = extendPotential(Potential,ZAxis)
    # plt.plot(ZAxis,Potential)

    # electron charge in C
    q = 1.602176634e-19

    z = ZAxis

    inds, props = sig.find_peaks(-Potential,width=10)

    z0 = z[inds[-20]]

    dz = z[1]-z[0]

    # Find middle of z-axis
    zmid = np.floor(len(z)/2)

    # Take the last minimum to be the surface
    zedge = inds[-2]+14

    sigma = 1 # in A

    # From OUTCAR. Ordered OH2, OH, O, OOH
    totalEnergyCond = np.array([2.0804+flt, 1.7528+flt, -
0.1106+flt, 1.5475+flt])*q
    totalEnergyVal = np.array([0.6702+flt, -0.3905+flt, -
0.6051+flt, -0.1430+flt])*q

    k0 = initialMomentum(totalEnergyVal[0], np.exp(-(z-
z0)**2/(2*sigma**2)), Potential*q, \
                         dz*1e-10, sigma, np.max(z), z0)
    k0 = np.abs(float(np.real(complex(k0[0]))/1e10))

    wavefun = np.exp(1j*k0*z)*np.exp(-(z-z0)**2/(2*sigma**2))
    owavefun = wavefun = np.exp(1j*k0*z)*np.exp(-(z-
z0)**2/(2*sigma**2))
    norm = np.linalg.norm(wavefun,ord=2)

    wavefun = wavefun / norm / np.sqrt(dz)

    Vexp = potentialExpectation(wavefun, Potential, dz)

    wavefuncs = np.zeros((n+1,z.size),dtype=np.cdouble)

    timesteps = timestep*np.arange(n)
    probcurs = np.zeros((n+1,z.size),dtype=np.cdouble)
    for i in range(n):
        wavefuncs[i,:] = wavefun
```

```
        probcurs[i,:] = calculateProbabilityCurrent(wavefun,
KAxis)
        cumprob[i,flt] = cumulativeProb(wavefun, z, z[zedge], dz)
        wavefun = RunOneTimestep(wavefun, timestep, Potential,
KAxis)


    headers = ['$*OH_2$','$*OH$','$*O$','$*OOH$']


    flt = flt + 1

# end of outer loop


figt = plt.figure()
pxlabel = 'Time t [fs]'
ylabel = 'Cumulative Probability Beyond Surface'
axist = plt.axes(xlim=(0,n*timestep),ylim=(0,1))
tline = axist.plot(timesteps,cumprob,linewidth=2)
plt.xlabel(pxlabel,fontsize=14)
plt.ylabel(ylabel,fontsize=14)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
#axist.legend([r'$*OH_2$',r'$*OH$',r'$*O$',r'$*OOH$'])
axist.legend(["Original Energy","+1 eV","+2 eV","+3
eV"],fontsize=14)



import tkinter as tk
import numpy as np
from tkinter import filedialog
from ReadLOCPOT import ReadLOCPOT
import matplotlib.pyplot as plt
import matplotlib.animation as anim
from CPFunctions import *
import scipy.signal as sig

root = tk.Tk()
root.withdraw()


n = 5000
timestep = 0.001 # in fs
flt = 0
cumprob = np.zeros((n,4),dtype=np.cdouble)

for fl in ['OHH', 'OH', 'O', 'OOH']:
    #FilePath = filedialog.askopenfilename(title="Choose LOCPOT
File")
    #FilePath = 'D:\OneDrive - Technion\Charge Transfer\LOCPOT-'
+ fl
    FilePath = 'G:\LOCPOT-' + fl
    Potential, ZAxis, KAxis = ReadLOCPOT(FilePath,5,5000,0)
    #Potential, ZAxis = extendPotential(Potential,ZAxis)
    # plt.plot(ZAxis,Potential)

    # electron charge in C
    q = 1.602176634e-19

    z = ZAxis
```

```
    inds, props = sig.find_peaks(-Potential,width=10)

    z0 = z[inds[-20]]

    dz = z[1]-z[0]

    # Find middle of z-axis
    zmid = np.floor(len(z)/2)

    # Take the last minimum to be the surface
    zedge = inds[-2]+14

    sigma = 1 # in A

    # From OUTCAR. Ordered OH2, OH, O, OOH
    totalEnergyCond = np.array([2.0804, 1.7528, -0.1106,
1.5475])*q
    totalEnergyVal = np.array([0.6702, -0.3905, -0.6051, -
0.1430])*q

    k0 = initialMomentum(totalEnergyVal[flt], np.exp(-(z-
z0)**2/(2*sigma**2)), Potential*q, \
                         dz*1e-10, sigma, np.max(z), z0)
    k0 = np.abs(float(np.real(complex(k0[0]))/1e10))

    wavefun = np.exp(1j*k0*z)*np.exp(-(z-z0)**2/(2*sigma**2))
    owavefun = wavefun = np.exp(1j*k0*z)*np.exp(-(z-
z0)**2/(2*sigma**2))
    norm = np.linalg.norm(wavefun,ord=2)

    wavefun = wavefun / norm / np.sqrt(dz)

    Vexp = potentialExpectation(wavefun, Potential, dz)

    wavefuncs = np.zeros((n+1,z.size),dtype=np.cdouble)

    timesteps = timestep*np.arange(n)
    probcurs = np.zeros((n+1,z.size),dtype=np.cdouble)
    for i in range(n):
        wavefuncs[i,:] = wavefun
        probcurs[i,:] = calculateProbabilityCurrent(wavefun,
KAxis)
        cumprob[i,flt] = cumulativeProb(wavefun, z, z[zedge], dz)
        wavefun = RunOneTimestep(wavefun, timestep, Potential,
KAxis)


    headers = ['$*OH_2$','$*OH$','$*O$','$*OOH$']

    flt = flt + 1

# end of outer loop


figt = plt.figure()
pxlabel = 'Time t [fs]'
#ptitle = 'Cumulative Probability Beyond Surface'
axist = plt.axes(xlim=(0,n*timestep),ylim=(0,1))
tline = axist.plot(timesteps,cumprob)
#plt.title(ptitle,fontsize=14)
```

```
plt.xlabel(pxlabel,fontsize=14)
plt.ylabel("Cumulative Probability Beyond Surface",fontsize=14)
plt.yticks(fontsize=14)
plt.xticks(fontsize=14)
axist.legend([r'$*OH_2$',r'$*OH$',r'$*O$',r'$*OOH$'],fontsize=14)
```