can also be changed via modifying the surface of the dots. As at the surface the atoms still have valent electrons left over, ligands can be attached. When this is done with for instance CdSe QD's, the type of ligands don't alter the QD properties too much and it mostly affects the interaction with other particles. A basic picture of these dots are the nanocrystal core with a ligand shell, due to the fact that here you don't have pure covalent bonds, which causes the ligands to not have a very big effect. However, when using QD's with pure covalent bonds, like in silicon, this idea changes. Here ligands have shown to do have significant effects, on for example the density of states, charge transfer and k-space.

2 Creating QD's With Python

In order to analyse the properties of QD's with the DFT simulation, an input file must be created. Such a (.xyz) file (example in figure 26) contains all atoms in the QD, their element and their coordinates. Software exists to create these QD's manually, but this can be tedious and opens the door to errors. As such, to create QD's where only the saturation of the added ligands varied between the particles a Python script was written to enable people to build QD's with a certain amount of input parameters. In this section it will be outlined how this script was developed in order to give an insight how the programme works and also enable others to add to the script later. As an illustration some snippets of the code have been screenshotted and added to the appendix. The full code is available on GitHub (`https://github.com/timok1/QD_bachelor_project`).

The script starts with the bare nanocrystal structure, without any connected ligands. A new crystal can be created, to some limited specifications, or an already existing crystal can be given as input. When building a new crystal the script will ask for two atomic elements making up the core crystal and the lattice constant. These elements can be the same, in which case the crystal will become a regular diamond cubic structure. The requested elements will be placed at their correct positions in the zincblende unit cell (figure 2a (visual), figure 15 (code)), after which the unit cell is copied in all directions to the specified diameter. Then it will make cutoffs at the (1 0 0), (1 1 0), and (1 1 1) planes, leaving the bare crystal that will be used (figure 2b). While building the crystal, the programme assigns identification (ID) numbers to all atoms, and it keeps track of which atoms are bonded to which by checking for all atoms within bonding range (code in figure 16). All of the relevant information is stored in a dictionary (figure 24). Dictionaries in Python are data structures consisting of key-value pairs, where values can be dictionaries as well. There is an option to leave in atoms which are bound to just one other atom, but in reality this is a very unstable bond and unlikely to happen.

Aside from creating the core with the script, there is also the option to use existing .xyz files. You can for instance build the core in Vesta (http://jp-minerals.org/vesta/), leaving the user with more freedom regarding the geometry of the crystal. In this case, the script reads the input .xyz file and obtains all relevant information, i.e. coordinates, elements, ID numbers and bonds and stores it in a dictionary as well (figure 17).

After creating the core, the script determines possible sites where ligands can be added. This is done by checking which atoms in the crystal are bonded to less than four, the required valency of the elements in the crystal, other atoms. The direction of these sites are given by vectors relative to the atom the ligands connect to. The vectors point to the the vertices of a tetrahedron, owing to the tetrahedral structure of the crystal. This tetrahedral structure is maintained at all times. The orientation of the vectors is determined by creating a "test tetrahedron", and aligning all vectors with the direction of the bonded atoms of the base atom. The remaining vectors (those not pointing to existing atoms) are then the sites where ligands can be attached (figure 18).

Next, the user will be asked which ligands he wants to add. A small ligand library was created to go with the programme (figure 27). It consists of mostly organic ligands, such as regular butyl (figure 2c) or decyl. These ligands are .xyz files themselves, and were built in Avogadro (https://avogadro.cc/), free molecule building software. This software is also were the visualisations of the QD's and ligands in this paper come from. Here the atoms were connected in the right way, and the optimize geometry option was used to 'relax' the ligands. To get the script to recognise how to attach the ligands they had to be aligned along the correct axis, and the 'connection atom' had to be centered at the origin.
These ligands can also be extended with other ligands in the library (figure 19). This is done by replacing the last atom in the base ligand with all the atoms in the extension, oriented in the right direction. This can go on indefinitely, and as such you can create ligands like $C_{10}H_{21}$-COO-Na, while maintaining only a small library (figure 3). The atoms in the resulting ligands will not be perfectly positioned at the transitions from one ligand to another, but as they tend to be located at the extremities of ligands this will be taken care of relatively easy when optimising the structure in Avogadro or the DFT simulations.
Then the script asks what the coverage should be. Coverage here means the fraction of atoms open to ligands that actually will have a ligand attached, as an atom will only ever have one (non-single atom) ligand attached to it in order to reduce strain on the dot. Multiple types of ligands can be added to one QD.
The script will then pick a random site for each ligand and try to place it at that position

by first giving it a random rotation about its own axis and then rotating the ligand so that it aligns with the site. If a ligand cannot be placed at a certain site due to it being to close to another ligand, or the crystal itself, it will rotate the ligand (around its own z-axis) by 0.2 radians and try again. Whether other ligands are too close is determined by calculating the distance from all atoms in the ligand to all other atoms in the dot, and checking whether this distance is shorter than the sum of the bonding distance between the two elements and the buffer, which is a user input. Once a full rotation is made and the ligand was unable to be placed, the script tries a different site connected to the same base atom. If it's also impossible to place it there a different site will be randomly picked. After all requested ligands have been placed the remaining sites will be capped with a single atom of valency 1 (to prevent charged QD's), determined by the user. As these single atoms have to be placed at all remaining sites, more possible orientations have to be considered for this element in order not to be too close to other atoms. If it is too close to another atom, the closest atom is determined (ignoring its own base atom) and the single atom ligand will be rotated in the opposite direction around the base atom by 0.1 radians, after which this process repeats until a suitable position has been found. When this is completed, the QD is done (figure 2e) and will be written to an .xyz file.

To be able to compare the properties of QD's with varying ligands it is important that the only difference between the particles are the the properties of the ligands (e.g. the length of the ligands). In order to achieve this extra information is stored for every ligand that has been placed (figure 25). This enables the script to replace the ligands of a created QD with different ligands attached to the same atoms and with the same orientation. This is done in two possible ways. The first way (figure 22) is to first build a single QD, and then asking the user whether he wants to replace the ligands. Then the programme scans through every atom in the atom dictionary, and checks whether it's a ligand and what type it is. For each new type the user is asked what he wants to replace it with. Then, using the extra ligand information, the new ligands are placed in the same way as before, after which the ligands can again be replaced. Another way of getting multiple QD's is by asking the user for multiple ligand 'bases' and extensions, after which the script builds all possible combinations (figure 23). So if for example the input for the bases is $C_{10}H_{21}$, C_9H_{19}, C_8H_{17} and the extensions input is COO+Na and CF_3, six QD's will be created. The process for this is the same as with the first option, it just happens automatically. For both options it is best to create the QD's with the biggest ligands first, as that leaves more space for later QD's. To facilitate this even more the buffer can be determined per QD, leaving more space at first, and then being less strict about the required distance.

When building a QD for DFT simulations it is important to place the atoms at the correct distance from each other. First off, because the only information stored in the .xyz file are the coordinates and the elements of atoms. For that reason the only way for software to determine the saturation of bonds is by judging the distance between the atoms (unsaturated bonds are longer than saturated ones). Secondly, as the DFT simulations try to find the minimum of energy in the QD's, the process is shortened a lot if the energy in the QD is already low beforehand, e.g. by having the right bond lengths. Therefore, the bond lengths between a number of atoms are stored in a csv file (figure 28). These values were obtained in Avogadro by connecting two elements and using the optimisation tool available in Avogadro and then measuring the distance. As the same optimisation tool was used later when preparing the QD's for the DFT simulation this was deemed accurate enough.

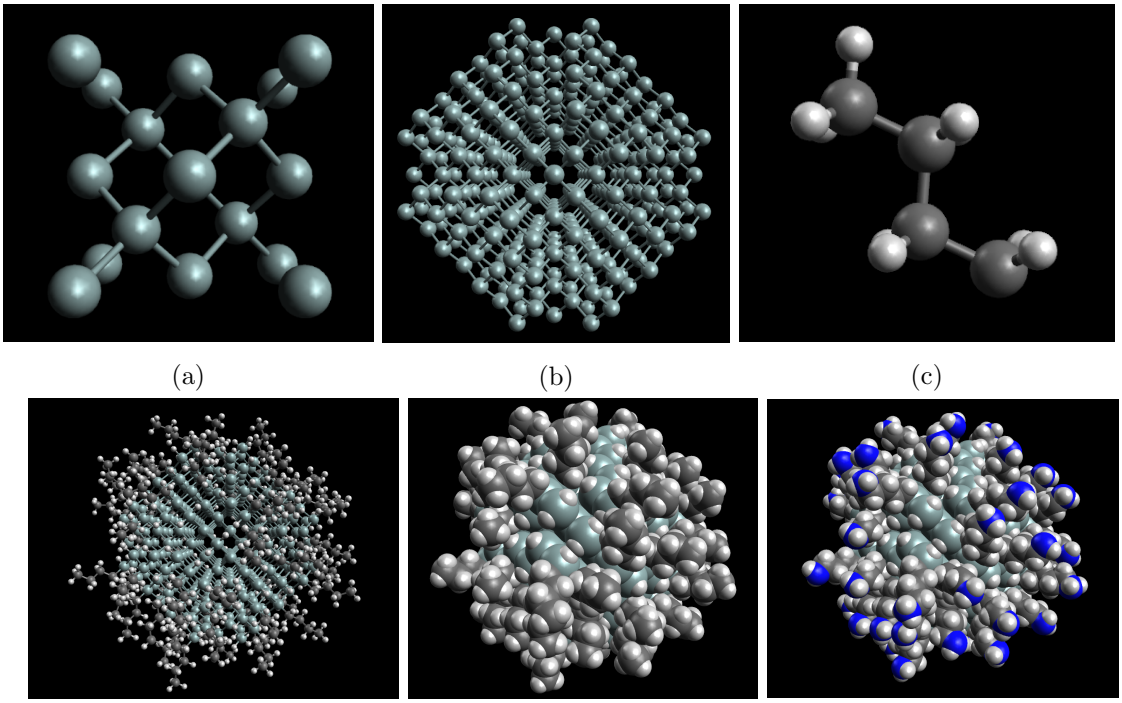

Figure 2: Process of creating a QD (a) The diamond cubic unit cell (b) Bare silicon core (c) Butyl ligand (d) Silicon QD with butyl ligands (e) Same crystal as in 2d, but showing Van der Waals spheres (f) Replaced ligands in 2d with C_4H_8 ligands extended with NH_2

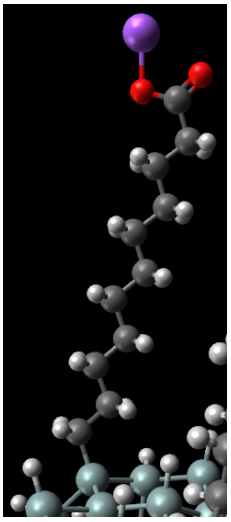

Figure 3: Close up of $C_{10}H_{20}$ ligand extended with COO and Na

3 Analysing QD's with DFT Simulations

Calculating the Schrödinger equation for many particles is difficult. For instance, QD4 has 1171 atoms and 6306 electrons, and all particles interact with each other. So even when using a supercomputer certain approximations have to be made in order to analyse this problem. First off, the Born-Oppenheimer approximation is used. In this approximation the wave functions of the electrons and the nucleons can be separated due to the fact that the mass of the nucleus is much larger than that of the electron, and the speeds of the electrons much higher. This results in an electron moving through a 'potential landscape' defined by the positions of the nucleons. Then, the electron-electron interaction needs to be looked at. Instead of looking at all these interactions individually, an electron is seen as moving through the field of all electrons together. Thus for every electron there are no longer 6305 electron-electron interactions, but only one. Additionally, pseudopotentials are used for (atomic) core electrons, as chemical and other interactions due to the electrons are mainly caused by valence electrons. Here the potentials are 'smoothed out' close to the core, and as a result the multiple nodes due to the orbitals disappear. At a certain distance from the core these pseudovalues are correct again (figure 4). Also the wave functions are replaced, often in the form of Gaussian functions around the core. There are multiple choices here, which can be more precise at the expense of the runtime of the simulations, like for all parameters. Here we have chosen the Double-Zeta basis set, where 2 functions are used for every orbital instead of just one. In carbon for instance, you would need five functions to describe the orbitals (one for 1s and 2s, and 3 in total for the $2p_x$, $2p_y$, $2p_z$ orbitals), and with Double-Zeta this is doubled to ten. The basis set was also chosen to be short-range. Because the quantum dots are semiconductors this