

Wandelen door kristallen

Blender als kristallografische visualisatietool

Jarrit BOONS

Promotor: prof. Christine E. A. Kirschhock Masterproef ingediend tot het behalen van
de graad van master of Science in de
industriële wetenschappen: E-ICT ICT

Promotor: Ann Phillips

Academiejaar 2018 - 2019

©Copyright KU Leuven

Zonder voorafgaande schriftelijke toestemming van zowel de promotor(en) als de auteur(s) is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden. Voor aanvragen i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, kan u zich richten tot KU Leuven Technologiecampus De Nayer, Jan De Nayerlaan 5, B-2860 Sint-Katelijne-Waver, +32 15 31 69 44 of via e-mail iiw.denayer@kuleuven.be.

Voorafgaande schriftelijke toestemming van de promotor(en) is eveneens vereist voor het aanwenden van de in deze masterproef beschreven (originele) methoden, producten, schakelingen en programma's voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.

Voorwoord

Deze pagina draag ik op aan een aantal mensen die me gesteund en geholpen hebben doorheen dit avontuur in de wereld van de kristallografie en Blender.

Eerst en vooral wil ik mijn promotor Chirstine Kirschhock bedanken voor al haar hulp, geduld en me te introduceren tot kristallografie. Zonder haar was dit onderzoek niet mogelijk geweest. Daarnaast betuig ik ook graag mijn dank aan Ann Philips, die steeds bereikbaar was om vragen te beantwoorden en deze tekst na te lezen.

Ook wil ik mijn ouders bedanken, die me hebben gesteund en geholpen waar mogelijk.

Ten slotte bedank ik graag mijn vriendin, Hanne, die steeds in me geloofde en, zelfs op de meest moeilijke momenten, altijd klaar stond om mijn moreel hoog te houden.

Our virtues and our failings are inseparable, like force and matter. When they separate, man is no more. - Nikola Tesla

Short summary

In the world of crystallography, which encompasses the study of crystals, it can sometimes be nearly impossible to comprehensively describe the structure of a crystal on paper, let alone in words, due to the sheer complexity of these structures. This is why researchers have been using software, which is able to visualise the structure of these crystals, to share their knowledge. Despite the amount of data these programs can already display nowadays, none of them have the full range of 3D virtual reality options current graphic software packages and game engines provide. In this thesis we have focused on one of these in particular, Blender.

Blender is an open source 3D computer graphic software tool which is mainly used for 3D modelling, creating animations and even as a game engine for 3D video games. Blenders makes use of it's own Blender Python API which is, like the software, free to use, and very well documented.

The main goal of this thesis was to firstly see whether it is possible to develop a program in Python that can visualise crystals in Blender. And how this compares to the crystallographic visualisation software which are currently used.

We divided this goal into three general parts: researching the state of the art and programs that might be helpful, actually developing the program and lastly testing the program and analysing the results.

To get a grasp on the structure of the crystals we want to visualise we explored the world of crystallography. We found that every crystal is build up out of unit cells. A unit cell is a 3D lattice in which the complete structure of a crystal resides. No matter how big a crystal is, it will always consist of a number of these identical unit cells.

The CIF format is a manner in which crystals can be described with ASCII characters. This makes it interpretable by both humans and computers and made an ideal format to use as input for our program. We used the CifFile module of PyCIFRW to parse these CIF files and convert the crystal data to more usable Python structures.

The second external program we used is OpenBabel. With OpenBabel we can calculate the positions of all the atoms within the unit cell from a given list of symmetry operation within the crystal.

To draw the unit cell of a crystal in Blender we needed to make use of the bpy module the Blender API offers. This module contains a lot of functions we used to both create and modify data in the Blender environment. The actual drawing of the crystal consists out of three steps: drawing the frame of the unit cell, drawing the atoms of the unit cell and lastly drawing bonds between

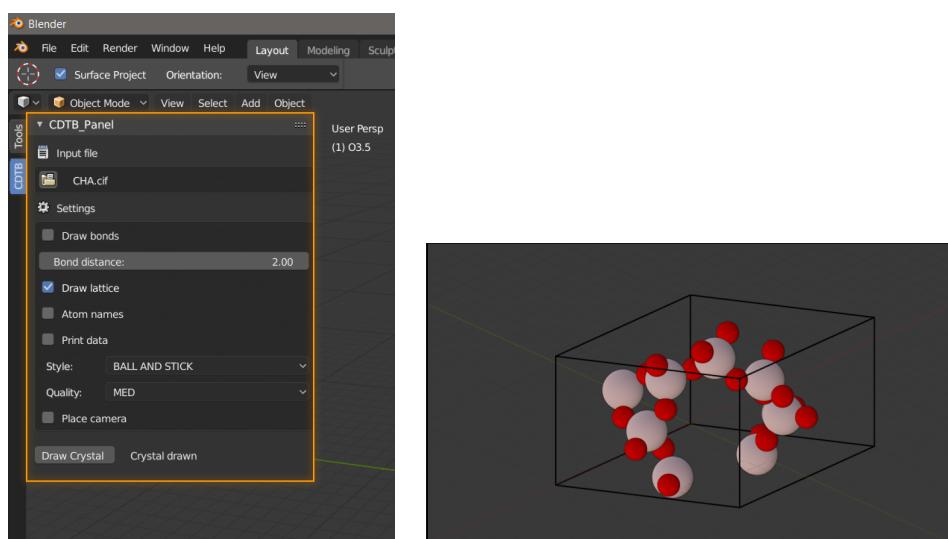
atoms. Atoms are drawn as spheres and have different radii and colours based on the element they represent. Whether or not a bond has to be drawn between two atoms, depends on the distance between them. The user has the option to choose this distance or can choose to not draw bonds.

To make our program easy to use, we presented our program in the form of a Blender add-on. Designing an add-on with the Blender API is done by creating a combination of operators classes, which run the functions of the program, and panel classes, with which we can customize the layout of the add-on.

To test our program we used three crystals with each a different number of atoms, and timed how long it takes for our program to draw these, with and without bonds. We found that for a smaller crystal, this takes about half a second, while for a larger crystal it can take up to two minutes to draw a crystal and its bonds.

When we compared these results with the time a current crystallographic program, VESTA, takes to draw a crystal, we found that our program takes about 16 times longer to draw a crystal with a small amount of atoms, while it takes up 400 times longer to draw larger crystals. This is mainly due to the fact that the bpy module is rather slow. About 80 percent of the total runtime of our program is used to draw the atoms which means that we only could optimise the remaining 20 percent of the program.

Despite the fact that our program is slower than VESTA, the crystal model it draws is more interactive and the program is both easier to use and more expandable than VESTA. Due to the nearly endless possibilities the combination of the Blender API and Python provide, it is very easy to add features to our program such as the ability to let the user define the boundaries between the crystal has to be drawn and using element specific bonding distances. This makes using Blender as a crystallographic visualisation tool a viable and future proof option and is definitely worth looking into more deeply.



Abstract

Nederlands

Omwille van de complexe structuren die kristallen soms hebben, is in kristallografie de vaardigheid om kristallen te visualiseren belangrijk voor het delen van kennis. De software die op dit moment gebruikt worden om kristallen te visualiseren is geavanceerd. Deze hebben echter niet de capaciteiten die hedendaagse grafische software en game-engines voorzien. Eén programma dat flexibel en krachtig is op het vlak van grafisch ontwerp is Blender. Blender is een open source 3D computergrafisch softwarepakket dat gebruikt wordt voor 3D modellering, animaties en als game-engine. In deze thesis onderzoeken we de mogelijkheid tot het gebruiken van Blender als kristallografische visualisatietool, door een interface te maken tussen het CIF-formaat en Blender, als basis voor toekomstige uitbreidingen. Door gebruik te maken van de Blender API en Python zijn we in staat een add-on te ontwerpen waarmee een gebruiker CIF-bestanden kan inlezen en eenheidscellen kan visualiseren. We merken echter dat het tekenen van kristallen in Blender lang duurt in vergelijking met andere kristalvisualisatiesoftware, zoals VESTA. We nemen waar dat het tekenen van eenheidscellen met een klein aantal atomen in Blender tot 14 maal langer duurt dan in VESTA, en voor eenheidscellen met een groot aantal atomen dit 400 maal zo lang duurt. We zien dat ongeveer 80% van de loopduur van het programma besteed wordt aan het tekenen van objecten. Dit wil echter niet zeggen dat Blender ongeschikt is om kristallen te visualiseren. De flexibiliteit van de Blender API, in combinatie met het groot aantal mogelijkheden dat Python biedt, maakt van Blender, hoewel een tragere, een krachtige kristallografische visualisatietool.

English

In crystallography, due to the complex structure crystals may have, the ability to visualise them is an important tool for sharing knowledge. The software that currently is used to visualise these crystals, already are quite advanced. However, none of these has the full range of 3D virtual reality options current graphics software and game engines provide. One program being extremely flexible and powerful in terms of graphical design is Blender. Blender is an open source 3D computer graphic software, and is mainly used for 3D modelling, animations and as a game engine. In this thesis we explore the possibility of using of Blender as a crystallographic visualisation tool by creating an interface between the CIF format and Blender as a base for further expansion. By using the Blender API and Python we are able to create an add-on with which a user can read in CIF files and draw unit cells in Blender. We find, however, that the drawing these crystals in Blender takes quite a while in comparison with other crystal visualisation tools, such as VESTA. We observe that for unit cells consisting of a rather small number of atoms, Blender takes about 14 times longer to draw this than VESTA, while for larger unit cells Blender can take up to 400 times longer to draw the unit cell. We find that around 80% of the total runtime of the program is spent on drawing the objects. However, this does not necessarily mean that Blender is not suited for visualising crystals. The flexibility of the Blender API in combination with the range of possibilities Python provides, makes Blender, although slower, a very powerful crystallographic visualisation tool.

Inhoudsopgave

Voorwoord	iii
Samenvatting	v
Abstract	vii
Inhoudsopgave	viii
Inhoud	ix
Lijst van figuren	x
Lijst van tabellen	xii
Lijst met afkortingen	xiii
1 Inleiding	1
1.1 Situering van het onderzoek	1
1.2 Probleemstelling	2
1.3 Doelstelling	2
1.4 Onderzoeksvragen	3
1.5 Structuur van de tekst	3
1.6 Conclusie	4
2 Literatuurstudie	5
2.1 Kristallografie	6
2.2 Het Crystallographic Information File (CIF) formaat	11
2.3 Kristalvisualisatiesoftware	14
2.4 CIF-Parsers	16
2.5 Blender en de Blender API	18
2.6 Conclusie	22

3 Algemeen ontwerpproces	24
3.1 Omvormen van het invoersbestand	24
3.2 Van CIF naar py	26
3.3 Tekenen in Blender	28
3.4 Creëren van een Blender add-on	29
3.5 Conclusie	33
4 Gedetailleerd ontwerpproces	34
4.1 Installatie van externe programma's	34
4.2 Inlezen van het bestand	36
4.3 Tekenen in Blender	40
4.4 Ontwerpen van een Blender add-on	47
4.5 Conclusie	51
5 Testen en Resultaten	54
5.1 Een kristal tekenen met de add-on	55
5.2 Resultaten	59
5.3 Vergelijking met VESTA	62
5.4 Valkuilen en problemen	64
5.5 Uitbreidingen	65
5.6 Conclusie	67
6 Besluit	69
Bibliografie	72

Lijst van figuren

2.1	Voorstelling van een eenheidscel in de roosterruimte (Borchardt-Ott, 2011)	7
2.2	De 6 kristalfamilies	8
2.3	De 14 Bravaisroosters en hun kristalsystemen	9
2.4	Twee elementen in een rooster met hun fractionale coördinaten	10
2.5	De user interface van VESTA3	15
2.6	De user interface van Crystalviewer9	15
2.7	De user interface van Olex ²	16
2.8	Poster van de film Next Gen (Tangent, 2018)	19
2.9	Het Blender startscherm	20
2.10	Het Blender project van de winnaar van een fotorealiteit wedstrijd, na rendering	21
3.1	Conversie van ruimtegroep met OpenBabel GUI	26
3.2	Vereenvoudigd klassendiagramma van het programma	27
3.3	De drie stappen in het tekenen van een kristal	29
3.4	Het paneel van de add-on in de Blender GUI	32
4.1	Voorbeeld van een bézierkromme met graad 3 (Tregoning, 2007)	45
4.2	Overzicht van de <i>hook modifier</i>	46
4.3	Eenvoudig voorbeeld van een <i>bevel</i>	47
4.4	Onze add-on afgebeeld in de <i>user.preferences</i>	48
4.5	Het paneel (onderdelen uit Listing[4.16] in rode kader)	51
5.1	Overzicht van Blendersversies op downloadpagina van blender	55
5.2	Openen van de <i>User Preferences</i> via GUI(links) en Zoekfunctie(rechts)	56
5.3	Duur van het programma in functie van gekozen kwaliteit	60
5.4	Vergelijking van de tekenkwaliteit voor het kristal OSO (van boven naar onder: MIN MED MAX)	62
5.5	Voorbeeld van kristal met foute bindingen (links) en correcte bindingen (rechts)	67

6.1 De add-on zoals te zien in Blender	70
--	----

Lijst van tabellen

2.1 Kristallen in wetenschappelijke takken	6
2.2 De submodules van de Blender bpy module (Conlan, 2017)	22
5.1 Overzicht van de instellingen van de add-on	58
5.2 Segmenten per atoom bij bepaalde kwaliteit	61
5.3 Vergelijking in tekenduur tussen ons programma en VESTA, tijd in seconden	63
5.4 Vergelijking duur van ons programma en het tekenen van enkel bollen, tijd in seconden	65

Lijst van afkortingen en begrippen

API

Application Programming Interface, een verzameling van definities en methodes

ASCII

American Standard Code for Information Interchange, standaard voor de codering van karakters

Bugfixes

Aanpassing aan een programma die fouten, of *bugs*, oplossen

IDE

integrated development environment, programma dat gemaakt is om software te ontwikkelen

GUI

Graphic User Interface, een visuele interface tussen een programma en de gebruiker

Interface

Manier van interactie tussen twee zaken

Open source

Een product dat vrij mag gebruikt worden

Syntax

Vorm en structuur van een tekst

XML

Extensible Markup Language, standaardformaat met syntaxregels voor het opslaan van gegevens

Hoofdstuk 1

Inleiding

Dit hoofdstuk omschrijft de context waarin deze thesis zich zal afspelen en vormt het fundament van deze scriptie. Om de rest van deze thesis te begrijpen is het van belang dit hoofdstuk grondig te lezen.

De eerste sectie van dit hoofdstuk schetst onder welke onderzoeks domeinen deze thesis kan worden ondergebracht. De tweede sectie beschrijft het probleem waarop deze thesis een oplossing zal trachten te bieden. Het algemene doel van deze thesis is het vinden en uitwerken van deze oplossing, dit doel zal verder worden opgedeeld in verschillende, kleinere, doelstellingen. In sectie vier worden deze doelstellingen geformuleerd als onderzoeksvragen bestaande uit een hoofdvraag en enkele deelvragen. Ten slotte is er nog een vijfde sectie waarin de hoofdstukken van deze scriptie worden beschreven. In een laatste sectie wordt dit hoofdstuk samengevat in een conclusie.

1.1 Situering van het onderzoek

Kristallografie is een tak van de wetenschap met als hoofdrol een eerder klein object, een kristal. Kristallen kunnen gezien worden als een puzzel van atomen. De stukjes van zo een puzzel kunnen verschillende chemische elementen zijn, maar evengoed allemaal dezelfde. Wat de kristallen zo uniek maakt, is hoe deze zijn opgebouwd. De opbouw bepaalt allerhande eigenschappen van dat bepaalde kristal. De complexiteit van kristallen kan oplopen tot op het punt waarop het bijna onmogelijk wordt deze in woorden te beschrijven en zelfs nog moeilijker deze te interpreteren. Dit is waar computertechnologie van pas komt.

Sinds enkele decennia zijn wetenschappers in staat kristalstructuren om te zetten in een digitaal formaat dat leesbaar is door zowel mensen als computers, het CIF-formaat (zie later). Dankzij dit standaardformaat is het mogelijk geworden de structuur van kristallen te lezen en te delen zonder kans op misinterpretatie, en biedt het computers de mogelijkheid zelfs de meest complexe kristalstructuren naar een driedimensionaal beeld om te zetten. Het 3D visualiseren van kristallen geeft wetenschappers meer inzicht en geeft meer mogelijkheden om hun kennis over te brengen. Dit alles heeft geleid tot een nauwe samenwerking tussen kristallografie en computerwetenschappen om de wereld van de kristallografie tot leven te brengen.

1.2 Probleemstelling

De programma's die op dit moment door wetenschappers worden gebruikt bij het 3D visualiseren van kristallen bieden nog lang niet de vrijheid en aanbod aan features welke sommige hedendaagse 3D software en game-engines te bieden hebben. Het van de grond opbouwen van 3D visualisatiesoftware of zelfs reeds bestaande software aanpassen is een enorme taak en slechts weinig wetenschappers hebben hier de tijd noch de technische knowhow voor. Daarnaast is het moeilijk om programmeurs te vinden die de nodige kennis bezitten over kristallografie en kristalstructuren of zich hierin willen verdiepen.

Het gebruiken van het tekenprogramma Blender om kristallen te visualiseren is een stap in de goede richting. Deze open source software laat, mits enige kennis van het programma, toe driedimensionale figuren te creëren en te bekijken. Het groot aantal features in Blender biedt de gebruiker veel vrijheid aan, wat ontbreekt in hedendaagse kristalvisualisatiesoftware. 3D objecten aanmaken en bekijken doet men via de grafische interface van Blender of met behulp van een script. Het gebruiken van scripts is erg interessant voor de gebruiker omdat deze hiermee langdurende of repetitieve taken kan laten uitvoeren.

In de context van kristallografie laat dit toe dat een kristal, bestaande uit een groot aantal atomen, niet meer manueel moet getekend worden. Helaas biedt dit geen volledige oplossing voor het probleem, voor elk kristal moet er nog steeds een apart script worden geschreven waarin alle informatie van dat kristal staat. Het scripten in Blender wordt gedaan aan de hand van Python en de API van Blender (zie verder). Hierdoor is het schrijven van scripts niet vanzelfsprekend en zonder enige voorkennis onbegonnen werk.

1.3 Doelstelling

Het algemene doel van dit werk is het ontwerpen van een interface die kristalstructuren kan visualiseren in het vooroemde open source programma, Blender. Eens dit bereikt is, zijn de mogelijkheden virtueel eindeloos.

Een van de problemen bij het visualiseren van een kristal met scripten in Blender is dat voor elke kristalstructuur een nieuw script moet worden geschreven. De interface moet gezien worden als een black box met als input de beschrijving van een kristalstructuur en de driedimensionale voorstelling hiervan als output. Dit stelt de nood aan een inputformaat dat interpreteerbaar is door een computer. In eerste instantie zal er enkel worden gezocht naar het meest praktische formaat, om de omvang van het programma te beperken. Dit kan nadien nog uitbereid worden zodat verschillende formaten kunnen worden ingelezen. De eerste doelstelling is dus een keuze te maken van het formaat dat er kan ingelezen worden door de interface.

De volgende stap in het ontwerpproces van de interface is het schrijven van een functie die in staat is het eerder gekozen formaat in te lezen en om te zetten naar bruikbare data. Afhankelijk van de complexiteit en striktheid van het formaat kan het ontwerpen van dit soort routines erg tijdrovend zijn. Het is mogelijk dit te vermijden door op zoek te gaan naar reeds bestaande Python modules met een gelijkaardige werking en deze, indien mogelijk, te implementeren in de interface. Eens

het formaat kan worden ingelezen, moet de nodige informatie worden opgeslagen. Python biedt de mogelijkheid om gebruik te maken van klassen, wat toelaat de kristaldata op te slaan in zelfgecreëerde datastructuren, wat de uiteindelijke dataverwerking zal vereenvoudigen.

De Blender API geeft de gebruiker een keuze uit een immens aantal functies. Dit zorgt er echter voor dat scripten in Blender zonder de nodige voorkennis erg complex kan worden. Een belangrijk onderdeel van deze scriptie is dan ook het zich verdiepen in de API van Blender en alle mogelijkheden die deze biedt. Met de nodige kennis van de functies wordt het mogelijk de gemaakte datastructuren driedimensionaal te visualiseren in Blender. Enkele basisfeatures die de interface moet hebben is een manier om elementen te onderscheiden aan de hand van hun kleur, automatisch bindingen maken tussen atomen op basis van hun onderlinge afstand en meerdere eenheidskristallen naast elkaar kunnen tekenen.

Ten slotte zullen de limieten van Blender getest worden in de context van het wetenschappelijk onderzoek rond kristallen.

1.4 Onderzoeks vragen

De doelstellingen uit vorige sectie kunnen worden geformuleerd als onderstaande onderzoeks vragen.

Hoofdvraag:

Is het mogelijk een interface te ontwerpen die in staat is kristalstructuren driedimensionaal te visualiseren in Blender?

Deelvragen:

Wat is de meest efficiënte methode om een kristalstructuur om te zetten in verwerkbare data?

Wat is de beste manier om kristaldata te visualiseren in Blender?

Is Blender een bruikbaar alternatief voor huidige kristalvisualisatiesoftware?

Als uitbreiding kan volgende deelvraag nog worden onderzocht:

Hoe kan het programma verder worden uitgebreid binnen de context van kristallografie?

1.5 Structuur van de tekst

Het eerste hoofdstuk geeft een inleiding tot het algemene onderwerp van deze thesis. Hier zal onder andere de probleemstelling en het doel van dit onderzoek worden besproken. Dit hoofdstuk zal een kort overzicht te geven over de verdere inhoud van deze scriptie.

In het tweede hoofdstuk wordt de literatuurstudie besproken. Dit onderdeel van de tekst verdiept zich in kristallografie, het gebruikte formaat van kristalbeschrijving, reeds bestaande visualisatie software, de parser en tot slot Blender en de Blender API. Het doel van dit hoofdstuk is inzicht geven in het theoretische aspect van de thesis en de nodige kennis verschaffen over de gebruikte technologieën.

In het derde hoofdstuk wordt het ontwerpproces beschreven. Hier kunnen de stappen worden gevolgd die zijn ondernomen in het opbouwen van de interface. Er wordt op een oppervlakkige manier

gekeken naar de opbouw van het programma om het globale overzicht te behouden. Dit hoofdstuk schetst de algemene structuur van het programma en de gedachtengang tijdens het ontwerpen hiervan.

De inhoud van het vierde hoofdstuk beschrijft de structuur van de ontworpen interface en hoe deze juist werkt. Deze tekst geeft een technische kijk op het programma en overloopt bepaalde onderdelen van de geschreven code in meer detail. Er wordt onder andere dieper ingegaan op de werking van de Blender API en de CIF-parser.

Hoofdstuk vijf kijkt in detail naar de output van het programma. Er wordt dieper ingegaan op de resultaten, complicaties, mijlpalen en gemaakte fouten. De eindconclusie van de thesis wordt besproken in hoofdstuk zes. Alle relevante informatie uit voorgaande hoofdstukken wordt hier opgesomd om een duidelijk overzicht te geven over het volledige onderzoek. De resultaten worden in dit hoofdstuk nogmaals kort overlopen om zo tot een uiteindelijk besluit te komen betreffende dit eindwerk.

1.6 Conclusie

Dit hoofdstuk gaf een overzicht over het onderwerp en de omvang van deze thesis, kaartte de probleemstelling aan en formuleerde enkele onderzoeks vragen die als rode draad doorheen deze tekst dienen. Met de kennis uit dit hoofdstuk zal de stap naar zowel de literatuurstudie als naar het meer praktische onderdeel van dit onderzoek minder groot zijn.

Hoofdstuk 2

Literatuurstudie

Vooraleer een oplossing kan worden ontworpen voor een probleem is het van belang de nodige informatie te verzamelen over de huidige staat van het probleem en eventuele reeds bestaande oplossingen. In dit hoofdstuk zal er onder andere worden gekeken naar de technologieën die op dit moment in de wetenschap gebruikt worden en welke technologieën mogelijk een oplossing kunnen bieden voor het probleem.

Het lezen van dit hoofdstuk is aangeraden aangezien het enkele belangrijke begrippen en technologieën beschrijft waarnaar zal worden verwezen in dit onderzoek. Het is echter mogelijk dit hoofdstuk over te slaan als deze informatie reeds gekend is of als de theorie achter het proces minder van belang is.

Omdat dit onderzoek zich deels afspeelt binnen de wereld van kristallografie, een specifieke tak van de wetenschap, zal de eerste sectie van dit hoofdstuk gewijd worden aan het overlopen van enkele begrippen die van belang zullen zijn in het verdere verloop van het onderzoek. Daar kristallografie een eerder complexe wetenschap is, zal deze tekst een vereenvoudigde beschrijving zijn. Voor een meer accurate beschrijving wordt er verwezen naar de literatuur waarop deze sectie gebaseerd is (Borchardt-Ott, 2011). De tweede sectie zal het Crystallographic Information File of CIF-formaat beschrijven. Dit tekstformaat ligt aan de basis van het digitaliseren van kristalstructuren en zal gebruikt worden bij het inlezen van kristaldata. In de derde sectie wordt gekeken naar welke programma's op dit moment worden gebruikt bij het visualiseren van kristallen en hoe deze werken. De vierde sectie kijkt naar enkele reeds bestaande CIF-parsers en of deze al dan niet kunnen gebruikt worden in dit onderzoek. Sectie vijf zal zich verdiepen in de werking van Blender en de Blender API. In de zesde en laatste sectie wordt een conclusie getrokken uit de verkregen informatie.

Tabel 2.1 Kristallen in wetenschappelijke takken

Biologie	Chemie	Farmacologie	Geologie
proteïnes	rubber	alle vaste medicijnen	alle mineralen
polysacharides	benzeen	vitamines	metalen
beenderen	naftaleen		

2.1 Kristallografie

2.1.1 Wat is kristallografie

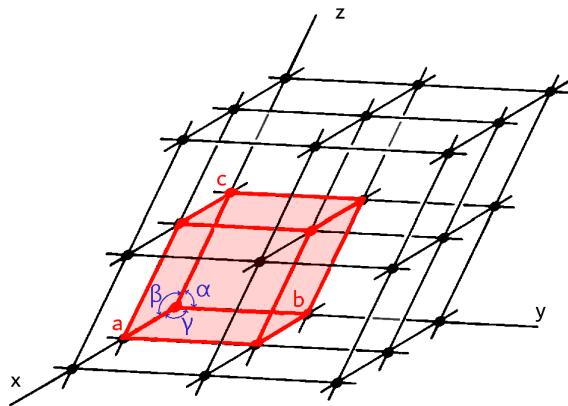
Kristallografie kan beschreven worden als de studie van materie in een kristallijne staat en houdt zich onder andere bezig met de synthese en opbouw van kristallen en hun fysische en chemische eigenschappen. Eerder in dit hoofdstuk werd kristallografie beschreven als een specifieke tak van de wetenschap, het is echter vermeldenswaardig dat dit een erg omvattende studie is. Kristallen komen namelijk voor in vrijwel elk ander wetenschappelijk domein. Tabel[2.1] toont enkele voorbeelden van kristallen en de respectievelijke tak van de wetenschap waar deze onder vallen.

In dit onderdeel zullen verder enkel de kristallografische begrippen worden beschreven die binnen de omvang van deze thesis vallen.

2.1.2 De eenheidscel

Alle kristallen zijn gedefinieerd als een periodische schikking van atomen, ionen of moleculen. Het opbouwen van een kristal wordt gedaan aan de hand van roosterpunten. Een aantal roosterpunten op een lijn met een gelijke onderlinge afstand wordt een roosterlijn genoemd, een verzameling van evenwijdige roosterlijnen met eenzelfde onderlinge afstand is een roostervlak. Ten slotte kan dit worden herhaald in een derde dimensie en wordt er een roosterruimte gevormd, welke in zwart getekend staat op Figuur[2.1]. Een roosterruimte kan in essentie zo groot zijn als het beschreven kristal. Zoals eerder ook gezegd is een kristal periodisch, dit wil zeggen dat er enkel moet gekeken worden naar het kleinste unieke volume van een kristal. Dit uniek volume valt volledig binnen de roosterruimte beschreven door de eerste acht roosterpunten. Het volledige kristal kan worden opgebouwd door de translatie van deze eenheidscel in 3 richtingen in de ruimte. Een kristal bestaat gewoonlijk uit een groot aantal van deze eenheidscellen.

Geometrisch gezien bevat een eenheidscel 3 paar evenwijdige parallelogrammen als omhullende vlakken en kan het beschreven worden aan de hand van 6 variabelen, welke de roosterparameters worden genoemd. De parameters a , b en c bepalen de lengte tussen de roosterpunten volgens de drie assen van de roosterruimte. De drie resterende variabelen worden gebruikt om de hoeken tussen deze assen te beschrijven en worden α , β en γ genoemd. Deze variabelen staan ook aangeduid op Figuur[2.1] met de eenheidscel in rood.

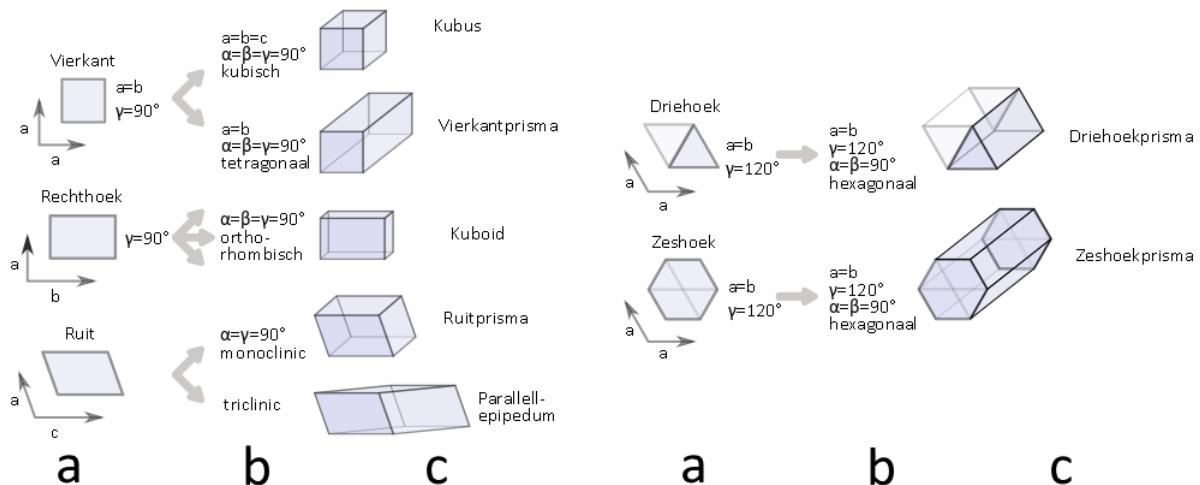


Figuur 2.1: Voorstelling van een eenheidscel in de roosterruimte (Borchardt-Ott, 2011)

2.1.3 Classificatie van kristalroosters

De vorm van de eenheidscel van een kristal is niet willekeurig, omdat dit volume de ruimte volledig moet kunnen vullen. Mogelijke types van eenheidscellen kunnen worden herleid op basis van de vorm van het vlak dat deze beschrijft, zie kolom a van Figuur[2.2]. De driedimensionale eenheidscel kan dan worden verkregen door de extrusie van dit vlak in de derde dimensie, zie kolom c van Figuur[2.2]. Zo bestaan er exact zes coördinaatsystemen, gebaseerd op de restricties van de roosterparameters. Deze vormen de basis voor de classificatie in kristalfamilies en krijgen de naam: kubisch, tetragonaal, orthorombisch, monoclinisch, triclinisch, en hexagonaal. Deze worden weergegeven in kolom b van Figuur[2.2].

Vertrekende van de mogelijke coördinaatsystemen en rekening houdend met mogelijke symmetrieën ontstaan er zeven kristalsystemen, zie kolom a van Figuur[2.3]. Hierbij wordt het hexagonale coördinatenstelsel verder opgesplitst in het trigonale en hexagonale kristalsysteem, afhankelijk van de aanwezigheid van een drie- of zesvoudige symmetrie. De zeven resulterende primitieve eenheidscellen beschrijven voor de zeven kristalsystemen alle mogelijke eenheidstranslaties in de richtingen van x, y en z. Dit wil zeggen dat een atoom of molecule op een positie (x,y,z) ook altijd aanwezig zal zijn in de richtingen van x, y en z als de afstand identiek is aan een veelvoud van de eenheidstranslaties in deze richtingen.



Figuur 2.2: De 6 kristalfamilies

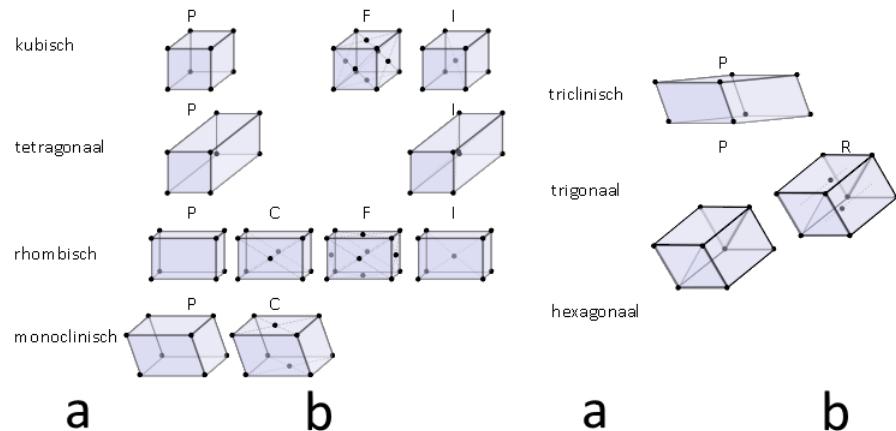
De Franse natuurkundige Bravais (Britannica, 2018) geeft een extra indeling op basis van mogelijke translaties binnen de eenheidscel, welke in overeenkomst met de symmetrieveroorwaarden van de kristalsystemen zijn. Dit leidt tot de 14 Bravaisroosters, ook wel gecentreerde roosters genoemd welke worden beschreven met een letter. De eerste wordt de primitieve genoemd en krijgt de letter P toegekend. Bij deze bestaan alleen de eenheidstranslaties. Het rooster bevat alleen roosterpunten op de 8 hoeken van de cel.

Het tweede gecentreerde rooster noemt men grondvlakgecentreerd. Deze heeft naast de eenheidstranslaties ook een centrering in een vlak. Elk deeltje kan ook teruggevonden worden over een translatie van een halve lengte langs twee assen. Er kan sprake zijn van A, B of C centrering als de verschuiving langs de diagonaal van respectievelijk het bc-, ac- of ab- vlak aanwezig is. Ondanks dit verschil worden ze niet als aparte gecentreerde roosters beschouwd.

Wanneer er extra roosterpunten liggen in elk van voorgaande vlakken spreekt men van een vlakgecentreerd of F-rooster.

Ten slotte kan er zich, naast de roosterpunten op de hoeken, ook een in het centrum van het rooster bevinden, in dit geval wordt er van een ruimtegecentreerd of I-rooster gesproken.

In het geval van het trigonale kristalsysteem bestaat er een speciale vorm van centrering in overeenkomst met de drievoudige symmetrie, welke R-centrering genoemd wordt. Kolom b van Figuur[2.3] geeft de 14 resulterende Bravaisroosters weer, dit zijn alle mogelijke vormen dat een kristal kan aannemen.



Figuur 2.3: De 14 Bravaisroosters en hun kristalsystemen

2.1.4 Ruimte- en puntgroepen

Dit concept is vrij complex en minder van belang voor dit onderzoek. Omdat gerelateerde termen in het verloop van deze tekst aan bod komen is het toch best deze beknopt toe te lichten. In deze literatuurstudie zal de exacte theorie hierachter niet in detail beschreven worden, maar enkel wat er van belang is voor het begrijpen van deze thesis.

Puntgroepen wijzen op de innerlijke symmetrie van een roosterpunt. Een puntgroep kan gezien worden als een verzameling van symmetrieoperaties. Een symmetrieoperatie wordt als een manipulatie van een object beschouwd welke het object in zijn oorspronkelijke vorm transformeert. Naast de identiteit van een object met zichzelf zijn er vier verschillende soorten van symmetrieoperaties: spiegeling in een vlak, rotatie rond een as, inversie en de combinatie van een spiegeling en een inversie, wat een draai-inversie wordt genoemd. In totaal bestaan er 32 kristallografische puntgroepen, welke in overeenkomst zijn met de symmetrieveroorwaarden van de 7 reeds besproken kristalsystemen.

Ruimtegroepen kijken niet enkel naar innerlijke symmetrie maar ook naar de symmetrie van de kristalstructuren en worden verkregen door de puntgroepen toe te passen op de 14 reeds gekende Bravaisroosters. Dit leidt tot in theorie 448 combinaties, maar door de gelijkenis tussen sommige is dit nummer terug te brengen naar een totaal van 230 ruimtegroepen. Dit heeft als gevolg dat er 230 verschillende manieren bestaan om deeltjes te ordenen in een eenheidscel. Als gevolg kan elk kristal altijd in een van deze ruimtegroepen beschreven worden.

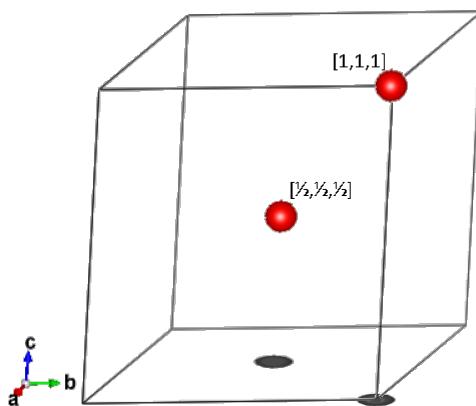
2.1.5 Beschrijving van een kristal

Met voorgaande informatie in het achterhoofd is het mogelijk een kristalstructuur te beschrijven. De enige informatie die hiervoor nodig is, is de ruimtegroep, het gebruikte coördinaatsysteem en de mogelijke centrering die alle symmetrieoperaties definieert. Vervolgens worden de roosterparameters van de eenheidscel gegeven en de lijst van elementen waarop deze symmetrieoperaties geldig

zijn. Ten slotte kan er ook een lijst worden gegeven van alle atomen die niet door het uitoefenen van de gegeven symmetrieoperaties geplaatst kunnen worden. Deze symmetrieonafhankelijke atomen in hun elementaire cel worden een asymmetrische eenheid genoemd. Met voorgaande gegevens kan elk mogelijk kristal afgebeeld en beschreven worden.

2.1.6 Het fractionele coördinatensysteem

De positie van atomen binnen het eenheidskristal wordt meestal beschreven aan de hand van zijn x-, y- en z-waarden. Deze waarden stellen steeds hun relatieve positie voor ten opzichte van de respectievelijk a- b- en c-waarden van de roosterparameters, vandaar de naam fractioneel, en liggen steeds tussen nul en een. Het gebruik van het fractionele systeem heeft als groot voordeel dat het erg eenvoudig te interpreteren is, zo hoeft er geen rekening gehouden te worden met de hoeken tussen de assen van het eenheidskristal. Een element met waarden $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$ ligt dus bijvoorbeeld in het midden van het kristal, terwijl een met waarden $(1,1,1)$ het hoekpunt van het rooster inneemt dat zich diagonaal tegenover de oorsprong van het rooster bevindt, Figuur[2.4].



Figuur 2.4: Twee elementen in een rooster met hun fractionale coördinaten

Het fractionele coördinatensysteem brengt natuurlijk ook enkele nadelen met zich mee. Om een kristal, en zijn elementen, weer te geven als een driedimensionaal figuur is er nood aan coördinaten volgens het orthogonale systeem, hiervoor kunnen de fractionele coördinaten dus niet gebruikt worden. Er zouden enkel waarden tussen nul en één kunnen verkregen worden wat telkens zou leiden tot een kubusvormig kristal. Deze foute dimensionering kan worden vermeden door deze te vermenigvuldigen met de lengte van hun respectievelijke as van het kristalrooster. Dit laatste biedt slechts deels een oplossing voor de omzetting van het fractioneel naar het orthogonaal coördinatensysteem en is enkel van toepassing voor kubische, tetragonale en orthorhombische kristalroosters aangezien deze uitsluitend bestaan uit rechte hoeken. Om een algemene formule op te stellen die gebruikt kan worden bij deze conversie moet er ook rekening gehouden worden met de hoeken tussen de assen van het kristalrooster. De algemene formule wordt voorgesteld als een matrix en ziet er voor een kristal met roosterparameters a, b, c, α , β en γ uit als volgt.

$$\begin{bmatrix} a & b \cdot \cos(\gamma) & c \cdot \cos(\beta) \\ 0 & b \cdot \sin(\gamma) & c \cdot \frac{\cos(\alpha) - \cos(\beta) \cdot \cos(\gamma)}{\sin(\gamma)} \\ 0 & 0 & \frac{\Omega}{a \cdot b \cdot \sin(\gamma)} \end{bmatrix}$$

Met Ω het volume van de eenheidscel.

$$\Omega = a \cdot b \cdot c \cdot \sqrt{1 - \cos^2(\alpha) - \cos^2(\beta) - \cos^2(\gamma) + 2 \cdot \cos(\alpha) * \cos(\beta) * \cos(\gamma)}$$

De orthogonale coördinaten van een element kunnen dan als volgt worden berekend:

$$\begin{bmatrix} x_{orth} \\ y_{orth} \\ z_{orth} \end{bmatrix} = \begin{bmatrix} a & b \cdot \cos(\gamma) & c \cdot \cos(\beta) \\ 0 & b \cdot \sin(\gamma) & c \cdot \frac{\cos(\alpha) - \cos(\beta) \cdot \cos(\gamma)}{\sin(\gamma)} \\ 0 & 0 & \frac{\Omega}{a \cdot b \cdot \sin(\gamma)} \end{bmatrix} \bullet \begin{bmatrix} x_{frac} \\ y_{frac} \\ z_{frac} \end{bmatrix}$$

Deze conversiematrix kan gebruikt worden voor elk element dat behoort tot dit kristalrooster en elk ander kristal met dezelfde roosterparameters.

2.2 Het Crystallographic Information File (CIF) formaat

2.2.1 Het STAR formaat

Om een beter idee te krijgen van de syntax van het CIF-formaat worden eerst de onderliggende concepten beschreven van het Self-Defining Text Archive and Retrieval (STAR) bestandsformaat, waarop het CIF-formaat gebaseerd is.

Het STAR-formaat was de eerste stap naar de digitalisering van verschillende soorten data. (Hall et al., 1991) Dit formaat bestaat uit ASCII tekst, wat makkelijke aanpassing en leesbaarheid toelaat door zowel mensen als computers. De opbouw van dit formaat ziet er uit als volgt: een file kan bestaan uit een of meerdere datablokken, welke nogmaals onderverdeeld zijn in een sequentie van data-elementen. De identiteit van een data- element wordt bepaald door een unieke naam die ervoor wordt geplaatst in het bestand. Het kleine aantal regels in een STAR-bestand maakt dit een eenvoudig en veelzijdig formaat. Zo zijn er geen restricties op de volgorde waarin de data moet worden genoteerd en moet er geen kennis zijn van het datatype van een element. Het gebruik van een lus maakt het mogelijk de toekenning van data- elementen te herhalen.

2.2.2 De CIF-notatie

Het CIF-formaat bouwt verder op de syntax van het reeds besproken STAR-formaat met een aantal extra voorwaarden. Enkel de datanamen die beschreven worden in de CIF-dictionary [Bijlage A]

worden toegelaten, dit woordenboek bevat dan ook alle parameternamen die nodig zijn om een kristal te beschrijven. De datanamen in het CIF-woordenboek zijn opgesteld door de Internationale Unie van Kristallografie (IUCr) en zijn dus algemeen aanvaard. Een lijn in het CIF-formaat mag niet langer zijn dan 80 karakters en een datanaam niet langer dan 32. Doordat in de CIF-dictionary datanamen worden opgedeeld op basis van het item dat ze beschrijven, is dit echter geen probleem. Net zoals bij de STAR-notatie is er geen verplichting tot het toekennen van datatypes, dit wordt bij het CIF-formaat echter wel aangeraden om de dataverwerking vlotter te laten verlopen. Zo worden data-elementen gezien als nummers wanneer ze beginnen met een cijfer. Een nummer kan een geheel getal, een kommagetal of als wetenschappelijke notatie worden gegeven. In het CIF-woordenboek worden ook de standaardeenheden van de data-elementen bijgehouden, de afmetingen van een eenheidscel worden verondersteld in Ångström te zijn. De Ångström is een eenheid die vaak gebruikt wordt in kristallografie en heeft de waarde van 0,1 nanometer. Data wordt beschouwd als tekst wanneer deze langer is dan één lijn. In het geval dat een data-element noch een nummer noch tekst is, wordt het beschouwd als een karakter.

2.2.3 Voorbeeld van een CIF-bestand

In dit onderdeel wordt aan de hand van listings uit een CIF-bestand de betekenis van de vaak voorkomende termen uitgelegd. Deze listings zijn slechts delen van het CIF-bestand van een bestaand kristal [Bijlage B] en dienen enkel als voorbeeld.

data_CHA

1

Listing 2.1:

De naam van het datablok.

```
# *****  
# CIF taken from the IZA-SC Database of Zeolite Structures 2  
# Ch. Baerlocher and L.B. McCusker 3  
# *****  
*****
```

Listing 2.2:

Alles achter een hashtag is commentaar en is enkel zichtbaar voor de lezer.

_cell_length_a	13.6750(0)	1
_cell_length_b	13.6750(0)	2
_cell_length_c	14.7670(0)	3
_cell_angle_alpha	90.0000(0)	4

_cell_angle_beta	90.0000(0)	5
_cell_angle_gamma	120.0000(0)	6

Listing 2.3:

De roosterparameters, de lengtes van de ribbes in Ångström en de hoeken tussen de assen in graden, een getal tussen haakjes achteraan wijst op de geschatte standaardafwijking van het getal.

_symmetry_space_group_name_H-M	'R -3 m'	1
_symmetry_Int_Tables_number	166	2
_symmetry_cell_setting	trigonal	3

Listing 2.4:

De notatie van de ruimtegroep, het nummer van de ruimtegroep en het kristalstelsel van de eenheidscel.

loop_		1
_symmetry_equiv_pos_as_xyz		2
'+x,+y,+z'		3
'2/3+x,1/3+y,1/3+z'		4
'1/3+x,2/3+y,2/3+z'		5

Listing 2.5:

Deze lus geeft de posities waarop de atoomgroep worden geplaatst, deze lijst is vaak erg lang.

loop_		1			
_atom_site_label		2			
_atom_site_type_symbol		3			
_atom_site_fract_x		4			
_atom_site_fract_y		5			
_atom_site_fract_z		6			
O1	O	0.9020	0.0980	0.1227	7
O2	O	0.9767	0.3101	0.1667	8
T1	Si	0.9997	0.2264	0.1051	9

Listing 2.6:

Deze lus beschrijft de groep van atomen en waar deze zich bevinden in het kristal. De eerste twee data-elementen bepalen respectievelijk het zelfgekozen symbool voor het element en de wetenschappelijke afkorting van het chemisch element. De drie laatste waarden geven de relatieve positie van deze atomen binnen de eenheidscel ten opzichte van de lengte van de ribben.

2.2.4 Kristallografische databanken

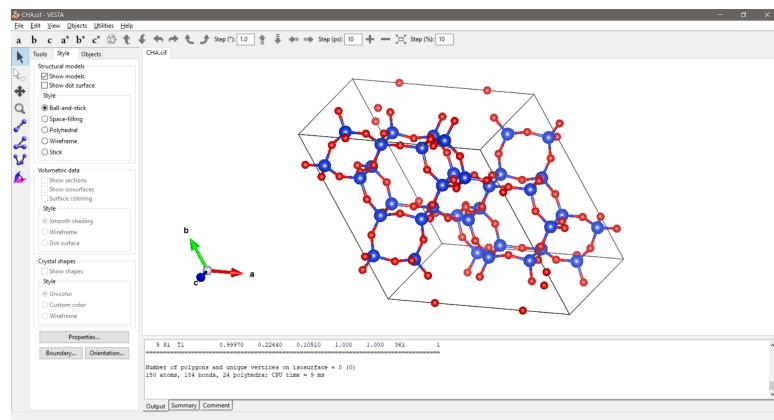
Aangezien het CIF-formaat gezien kan worden als een standaardformaat voor het beschrijven van kristallen bestaan er reeds CIF-bestanden voor vrijwel alle bestaande kristalstructuren. De ver-

zameling van deze bestanden, alsook andere formaten, worden kristallografische databanken genoemd en staan online ter beschikking op verschillende websites. De lijst met kristalstructuren blijft groeien en wordt bijgewerkt door verschillende organisaties en universiteiten. Een voorbeeld hiervan is de Crystallography Open Database (COD) waar de kristalstructuren van onder andere organische, anorganische en metaalorganische samenstellingen kunnen teruggevonden worden. Het voorbeeldbestand in vorige sectie is een deel van de beschrijving van de kristalstructuur van het zeolietyltype Chabaziet, dit CIF-bestand kan teruggevonden worden in de zeolietylstructuurdatabase op de website van de International Zeolite Association (IZA) welke zich uitsluitend bezighoudt met het onderzoeken en beschrijven van de verschillende types van het mineraal zeolietyl.

2.3 Kristalvisualisatie software

2.3.1 Visualization for Electronic and Structural Analysis (VESTA) 3

Als een van de meest frequent gebruikte 3D visualisatieprogramma's biedt VESTA 3 een grotere verscheidenheid aan features en voordelen dan zijn voorgangers en andere, gelijkaardige programma's doen. VESTA beschrijft zichzelf als een gratis te gebruiken 3D visualisatiesysteem voor structurele modellen, volumetrische data en kristalmorfologieën. (Momma and Izumi, 2008) De software wordt ondersteund op Windows, Mac en Linux. Geschreven in de programmeertaal C met het gebruik van de OpenGL technologie en een modern C++ GUI framework biedt VESTA een gebruikersvriendelijke interface met de mogelijkheid tot het roteren, translaten en schalen van het getekende object alsook het hierop plaatsen van fysieke gegevens zoals elektronendichtheden, nucleaire dichtheden, golffuncties en elektrostatische potentialen. Hiernaast bestaan er verschillende manieren om het object weer te geven. De omvang van deze objecten is in theorie oneindig en wordt slechts beperkt door de hardware van de computer, hoe groter het grafische rekenvermogen van deze hoe vlotter het zal functioneren. De grafische interface van VESTA wordt weergegeven op Figuur[2.5]. VESTA is in staat verschillende bestandsformaten om te zetten naar een driedimensionale voorstelling, zoals het CIF-formaat. VESTA laat toe nieuwe modellen aan te maken of bestaande modellen aan te passen. Gecreëerde objecten kunnen ten slotte geëxporteerd worden in de meest voorkomende 3D formaten zodat ze in andere 3D software, zoals Blender, kunnen worden bekeken.

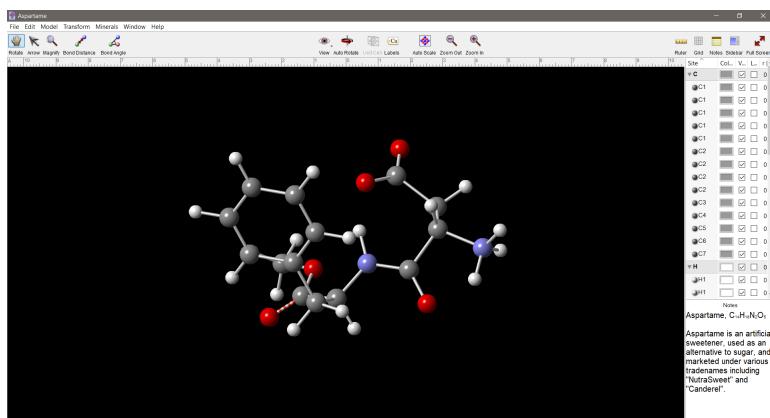


Figuur 2.5: De user interface van VESTA3

Omdat de broncode van VESTA niet openbaar beschikbaar is, is het vrijwel onmogelijk eigen features toe te voegen, en hangt de gebruiker vast aan de tools die deze software ter beschikking stelt. Deze beperking aan vrijheid heeft ertoe geleid dat wetenschappers op zoek gaan naar andere software die dit wel aanbiedt.

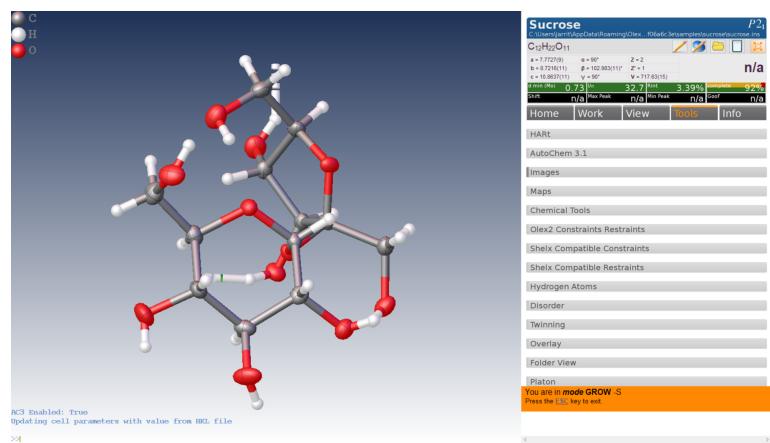
2.3.2 Andere 3D visualisatiesoftware

Naast VESTA bestaat er nog een groot aanbod aan andere software die in staat zijn kristallen driedimensionaal te visualiseren. Het Crystalmaker softwarepakket biedt het programma Crystalviewer, Figuur[2.6], gratis aan voor persoonlijk gebruik. Het strak design en de gebruiksvriendelijke user interface geven de software een luxueus gevoel. Voor het gebruik van het volledig potentieel van dit programma en om toegang te krijgen tot de documentatie wordt de gebruiker ertoe gedwongen het eerder dure, volledige softwarepakket aan te kopen. Hiernaast biedt de gratis versie niet de mogelijkheid andere formaten in te lezen dan het .crystal – formaat, wat niet gezien wordt als standaardformaat en niet bestaat in de kristallografische databanken.



Figuur 2.6: De user interface van Crystalviewer9

Een andere bekende tegenhanger van VESTA is Olex², zie Figuur[2.7]. Olex² biedt, net zoals VESTA, een groot aantal features aan, welke duidelijk beschreven worden in een overzichtelijke documentatie. Het softwarepakket is volledig gratis en ter beschikking van iedereen. Olex² is volledig geschreven in Python en maakt gebruik van enkele bibliotheken die de gebruiker de mogelijkheid biedt verschillende kristalbeschrijvingsformaten, waaronder CIF, in te lezen en te visualiseren. Ondanks de features en documentatie is het gebruiken van Olex² minder eenvoudig dan de eerder gezien programma's. Bij het testen van dit programma trad er ook een probleem op bij het visualiseren van een CIF-bestand, wat deze software voor deze toepassing vrijwel onbruikbaar maakt.



Figuur 2.7: De user interface van Olex²

2.4 CIF-Parsers

2.4.1 Wat is een parser

Een parser wordt gezien als een computerprogramma of script dat in staat is een input om te zetten naar een datastructuur met als voorwaarden dat deze input volgens een bepaalde structuur, denk XML, is ingegeven. De parser gaat in de ingevoerde gegevens op zoek naar herkenbare elementen en slaat deze op volgens een eerder genoemde datastructuur, bijvoorbeeld een boomstructuur of een dictionary. Sommige parsers worden gebruikt om ingegeven data te visualiseren, deze data kan dan worden gevisualiseerd in de vorm van een boomstructuur of een grafiek. Een ander soort van parsers heeft als nut de gegevens van het bestand om te zetten in een dictionary, waardoor deze in een ander programma eenvoudig kunnen worden opgevraagd. In dit onderzoek is er nood aan dit tweede type van parser, zodat de gegevens in een CIF-bestand kunnen worden omgezet naar door de module leesbare data.

2.4.2 Het schrijven van een parser

Om een parser te schrijven voor een bepaald bestandstype of formaat is er eerst een goede kennis nodig van dit formaat. Hoewel het op eerste zicht misschien niet logisch lijkt, wordt het schrijven van een parser eenvoudiger naarmate de striktheid van het formaat stijgt. Er moet minder rekening worden gehouden met alle mogelijke variaties die in het bestand kunnen voorkomen. Zoals in sectie twee besproken werd, bestaan er in het CIF-formaat weinig regels op vlak van tekststructuur. Hierdoor vergt het schrijven van een CIF-parser veel werk. In dit onderzoek is er aanvankelijk geprobeerd een eigen parser te ontwerpen die kan gebruikt worden om CIF-bestanden om te zetten naar bruikbare data. Dit proces wordt in hoofdstuk drie in meer detail besproken. Omdat dit echter niet het doel is van dit onderzoek, is er besloten te kijken naar de mogelijkheden die reeds bestaande CIF-parsers kunnen bieden. De bekeken parsers worden verder beschreven.

2.4.3 The Computational Crystallography Toolbox (cctbx)

Dit project heeft als voornaamste doel een brug te bouwen tussen kristallografische data en het gebruik hiervan in computer gerelateerde toepassingen. Het project is volledig open source, wat wil zeggen dat het volledig toegankelijk en gratis is voor iedereen die het wil gebruiken, hiernaast zorgt dit ervoor dat dit project voortdurend wordt verbeterd en stijgt in functionaliteit. Deze toolbox ligt ook aan de basis van de visualisatiesoftware Olex², welke beschreven wordt in de derde sectie van dit hoofdstuk. De grote omvang van dit project en het grote aantal functionaliteiten maakt het, ondanks de duidelijke documentatie, ook meer ingewikkeld om dit te gebruiken.

2.4.4 Python CIF Read/Write (PyCIFRW)

Naast het inlezen van CIF-bestanden kunnen deze bestanden met dit programma ook gecreëerd en aangepast worden. Ook dit programma is open source, omdat dit op een kleinere schaal gebeurt dan cctbx, zie vorige, zijn de mogelijkheden, hoewel meer beperkt, geschikter voor gebruik in dit onderzoek. Door de goede documentatie van dit project en de volledige ondersteuning van Python kan dit programma gecombineerd worden met de, in de volgende sectie besproken, Blender API. (IUCR, 2009) Voor deze thesis is de schrijffunctie minder van belang, deze zal niet worden bekeken.

```
import CifFile  
#Het inlezen van een CIF-bestand in een variabele wordt gedaan met de  
#functie:  
ciffile = CifFile.ReadCif("bestandsnaam.cif")  
#Vervolgens kunnen datablokken worden ingelezen:  
datablok = cf["een datablok"]  
#Een data-element uit het datablok kan worden gevonden met  
data_element = datablok["een datanaam"]  
#Of kan rechtstreeks worden aangesproken met
```

```
data_element = cf[ "een datablok "][ "een datanaam "]          10
#De data uit een lusblok van het datablok kan worden opgevraagd met
lb = datablok.GetLoop( "een datanaam uit de loop " )           11
12
```

Listing 2.7: Werken met PyCIFRW in Python3

Listing[2.7] is een voorbeeld van hoe PyCIFRW kan worden gebruikt in Python om een CIF-file in te lezen. De voornaamste functies worden hier weergegeven. Een meer gedetailleerde beschrijving van de werking van PyCIFRW kan worden teruggevonden in hoofdstuk 4.

2.5 Blender en de Blender API

2.5.1 Gratis software

Blender beschrijft zichzelf als een „open-sourced, community development program”, en is het collectieve werk van softwareontwikkelaars overal ter wereld. (Chronister, 2011) De ontwikkeling en het correct functioneren van dit programma wordt overzien door The Blender Foundation, een Nederlandse, onafhankelijke, non-profit stichting. Dit alles leidt ertoe dat de Blender software gratis verkrijgbaar is voor iedereen die het wenst te gebruiken.

2.5.2 Blender in de bedrijfswereld

Zoals in het verdere verloop van deze sectie duidelijk zal worden, heeft Blender een groot aantal verschillende toepassingen. Hierdoor wordt Blender niet enkel door amateurs gebruikt, maar ook door bedrijven in verschillende industrieën waaronder verkoop, manufacturing, game development en vele anderen. Een voorbeeld van de kracht van Blender is Tangent Animation, een animatiestudio die uitsluitend werkt met Blender. In 2018 ontwikkelde dit bedrijf de animatiefilm Next Gen welke te zien is op onder andere Netflix, Figuur[2.8].



Figuur 2.8: Poster van de film Next Gen (Tangent, 2018)

2.5.3 De Blender interface

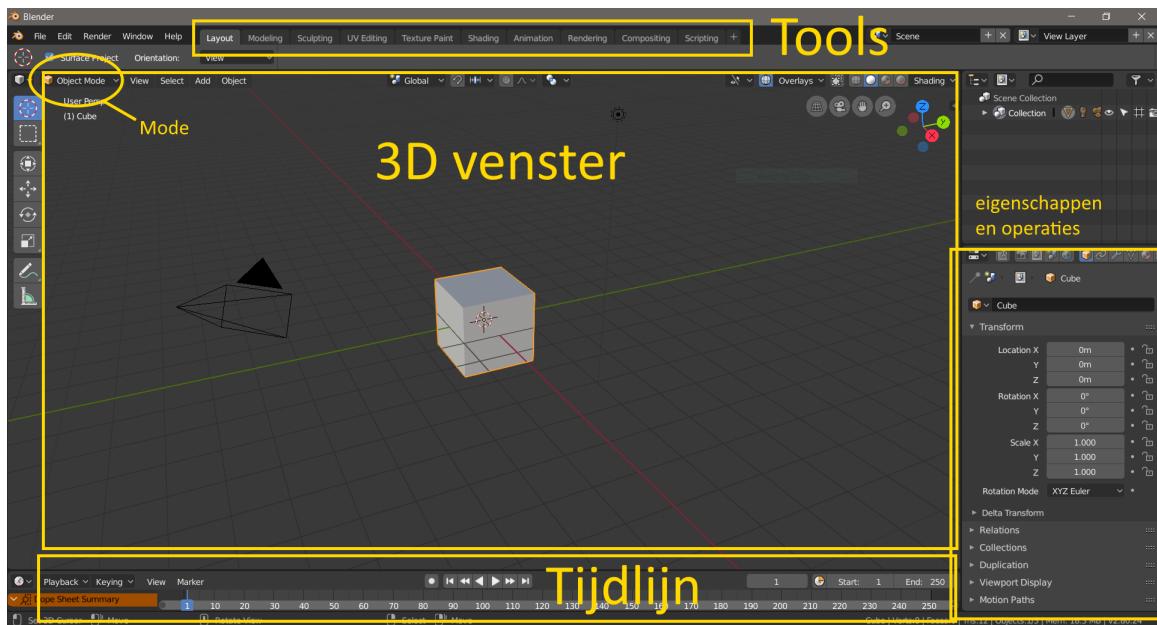
Na het opstarten van Blender wordt de gebruiker begroet met de Blender interface. Een niet ervaren gebruiker zal, naast onder de indruk, vooral in de war zijn en niet weten waar te beginnen. In Figuur[2.9] wordt het startscherm weergegeven en worden enkele onderdelen ervan beschreven. Veel van deze tools en hoe deze werken vallen niet binnen het bestek van deze tekst en worden weinig of niet uitgelegd, hiervoor wordt verwezen naar online handleidingen en de documenten waar deze tekst op gebaseerd is.(Chronister, 2011)(Conlan, 2017)

In het midden van het scherm kan het 3D venster worden teruggevonden. Hier worden alle objecten weergegeven die zich op de huidige ingeschakelde layers bevinden. Het gebruik van layers laat toe objecten op een andere layer tijdelijk te verbergen om zo een duidelijk overzicht te behouden over het huidige werk. Wanneer een object wordt aangemaakt zal deze automatisch geplaatst worden op de huidige layer. De Blender interface heeft twee modi: object mode en edit mode.

In object mode kan een object in zijn geheel bewerkt worden. Transleren, roteren en herschalen zijn slechts enkele van het groot aantal objectoperaties dat Blender aanbiedt.

Edit mode laat de gebruiker onder andere toe de vorm van het object aan te passen door hoekpunten aan te duiden en te verplaatsen. Objecten toevoegen wordt gedaan met de Add knop linksboven in het venster. In Blender gebruikt men de rechtermuisknop om items te selecteren en te verplaatsen. Het klikken op de linkermuisknop verplaatst de 3D cursor, dit is de plaats waar nieuwe objecten terechtkomen wanneer ze worden aangemaakt. Ten slotte kan in het 3D venster worden bewogen door het indrukken van het muiswiel.

Met deze informatie kan er een object aangemaakt en verplaatst worden en kunnen er enkele basisoperaties op worden toegepast. Dit is voorlopig alles wat er over het bewerken van objecten



Figuur 2.9: Het Blender startscherm

gezegd zal worden. Dit is slechts het topje van de enorme ijsberg die Blender wordt genoemd.

2.5.4 Kleuren, texturen en materialen

Blender laat de gebruiker toe de gemaakte objecten, naast vorm, ook kleur te geven. Eerst krijgt het object een materiaal toegekend, waarna de eigenschappen van dit materiaal kunnen worden aangepast. Deze eigenschappen zijn onder andere de kleur, lichtuitstraling, weerspiegeling, schaduw en doorzichtigheid van het object. Om het object er nog realistischer te laten uitzien kan er een textuur op het object worden gezet. Deze geven het object een bepaalde oppervlaktestructuur of uiterlijk. Blender voorziet een aantal basistexturen maar de gebruiker kan zelf afbeeldingen gebruiken om het voorwerp een textuur te geven. Wanneer de objecten een kleur of textuur hebben kan de gebruiker deze renderen. Dit gaat alle lichtinteracties en schaduws berekenen, wat erg computerintensief kan zijn. In Figuur[2.10] wordt de uitkomst van het renderen van een Blender project weergegeven, door slim gebruik van texturen en materiaaleigenschappen was de ontwerper in staat een 3D tekening levensrecht te laten lijken.(Blenderguru, 2012)

2.5.5 Andere opmerkelijke tools in Blender

Naast het aanmaken, vervormen en inkleuren van objecten biedt Blender nog een handvol extra mogelijkheden aan. Met de Sculpting tool kan men objecten boetseren tot in de kleinste details. Dit kan leiden tot realistische beeldhouwwerken.

Onderaan de interface afgebeeld op Figuur[2.9] kan het tijdlijn-venster gezien worden. Dit venster behoort tot de animation tool van Blender, door een aantal gerenderde frames snel opeenvolgend



Figuur 2.10: Het Blender project van de winnaar van een fotorealiteit wedstrijd, na rendering

achter elkaar te laten afspelen wordt een video verkregen. Kleine verschillen tussen deze frames geven de indruk dat het object beweegt, dit wordt een animatie genoemd.

Hoewel er betere software bestaat om dit te doen, is het in Blender, door gebruik te maken van scripts, mogelijk volledig functionele 3D videogames te ontwerpen. Zo bestaan er reeds games die volledig in Blender zijn gemaakt en te koop staan op Steam, een bekende videogames-distributeur.

2.5.6 Scripten in Blender met de Blender API

Nog een belangrijke functie die Blender aanbiedt, en in de vorige sectie niet werd vermeld, is de mogelijkheid tot het scripten in Blender zelf, in tegenstelling tot de game-engine. Omdat deze feature in zekere mate de rode draad is van deze thesis, wordt er een volledige subsectie gewijd aan dit eerder ingewikkelde onderdeel van Blender.

In de tweede sectie van dit hoofdstuk werd besproken hoe operaties in Blender kunnen uitgevoerd worden met behulp van de Blender interface. Er bestaat echter een andere manier om operaties uit te voeren, namelijk via scripts. Met de juiste kennis kan er met behulp van scripts zelfs meer gedaan worden dan met enkel de interface. Het schrijven van scripts wordt gedaan aan de hand van Python 3, verder gerefereerd als Python. Alle standaardmodules in Python kunnen dan ook gebruikt worden, hiernaast is het mogelijk zelf modules te importen, in het geval van dit onderzoek een CIF-Parser. Aan de hand van de Python logica is het eenvoudig loops te creëren die automa-

tisch een aantal objecten aanmaakt in Blender. Hieronder wordt kort de Blender API beschreven. De Blender API biedt een aantal modules aan die gebruikt kunnen worden, naast alle reeds bestaande functionaliteiten van Python. De voornaamste van deze modules is de bpy module, die alle functies bevat in verband met het aanmaken en aanpassen van objecten in Blender. De bpy module is zodanig groot dat deze nogmaals wordt opgedeeld in submodules, deze worden samen met hun taak weergegeven in tabel[2.2]. In de documentatie van de Blender API worden alle methodes van deze modules uitgelegd. In hoofdstuk vier wordt dieper ingegaan op het gebruik van de Blender API en de toepassing ervan in dit onderzoek.

Tabel 2.2 De submodules van de Blender bpy module (Conlan, 2017)

bpy.ops	Bevat alle operators voor het maken en aanpassen van objecten
bpy.context	Geeft de mogelijkheid specifieke data op te vragen
bpy.data	Bevat alle data van de objecten
bpy.app	Hulpmodule bij het schrijven van add-ons en extra functionaliteiten
bpy.types,bpy.utils,bpy.props	Hulpmodules bij het schrijven van add-ons
bpy.path	Vrijwel identiek aan de os.path submodule van Python

2.6 Conclusie

Kristallografie is een specifieke tak in de wetenschap die zich bezighoudt met het onderzoeken van kristallen, hun structuur en hun eigenschappen. Een kristal wordt opgebouwd uit eenheidscellen. Het beschrijven van zo'n eenheidscel geeft alle informatie van het kristal. De vorm van een eenheidscel wordt bepaald door zes roosterparameters en een van de 14 Bravaisroosters. Deeltjes kunnen zich op 240 manieren in de eenheidscel bevinden, de ruimtegroepen genaamd.

In het CIF formaat is alle informatie over een kristalstructuur terug te vinden, het is leesbaar door zowel computers als mensen. CIF bestanden worden geschreven in ASCII wat eenvoudige leesbaarheid en aanpasbaarheid toelaten. Gebaseerd op het STAR formaat heeft het CIF formaat enkele syntactische restricties waaraan moet gehouden worden. De structuur van CIF bevat datablokken, data-elementen en lussen om de data weer te geven.

De bestaande 3D kristalvisualisatiepakketten zijn vaak moeilijk in omgang, duur of bieden niet genoeg vrijheid. VESTA en Olex² zijn gratis programma's die kristalstructuren kunnen visualiseren en geven de gebruiker een aantal handige tools om onderzoek te vereenvoudigen. Ze hebben echter hun nadelen wat leidt tot de vraag naar andere programma's.

Parsers hebben als functie data te extraheren uit bestanden met een bepaalde syntax. Ze worden gebruikt om data te visualiseren of bruikbaar te maken in andere toepassingen. De complexiteit van het schrijven van een parser is omgekeerd evenredig met het aantal syntaxregels van een formaat, hierdoor is het schrijven van een CIF parser eerder moeilijk. Cctbx en PyCIFRW zijn open source modules die in staat zijn CIF bestanden te parsen.

Blender is een gratis, open source project waarin driedimensionale tekeningen kunnen worden gemaakt. Met een groot aantal tools, waaronder animeren, sculpten en een eigen game-engine, geeft Blender de gebruiker veel vrijheid en mogelijke toepassingen. Door gebruik te maken van texturen en materialen kunnen er fotorealistische tekeningen gemaakt worden. Scripten in Blender laat het automatiseren van tekenen toe en wordt gedaan aan de hand van Python scripts en de modules uit de Blender API.

Hoofdstuk 3

Algemeen ontwerpproces

In dit hoofdstuk worden de ondernomen stappen besproken bij het ontwerpen van een interface die kristalstructuren, in de vorm van een CIF-bestand, kan visualiseren in Blender. Dit hoofdstuk is met opzet kort en relatief oppervlakkig gebleven zodat de lezer de werking van het programma op een overzichtelijke manier kan doornemen. Een meer gedetailleerde beschrijving van dit proces kan worden gevonden in hoofdstuk vier, hier wordt ook dieper ingegaan op de installatie van de twee externe programma's, OpenBabel en PyCIFRW, die zullen besproken worden in dit hoofdstuk.

De eerste sectie beschrijft hoe de inwendige symmetrie een probleem vormt bij het visualiseren van kristallen, wat het programma OpenBabel doet en hoe het gebruiken hiervan een oplossing biedt op dit probleem. In de tweede sectie zullen de datastructuren worden uitgelegd die worden gebruikt en hoe, met behulp van een parser, een CIF-bestand kan worden omgezet in verwerkbare data. Ten slotte wordt verteld hoe de verkregen kristaldata zal worden opgeslagen in deze datastructuren. Het visualiseren van de kristaldata in Blender zal worden besproken in sectie drie. In deze sectie zal worden uitgelegd hoe de omkadering van het eenheidskristal, de atomen en hun onderlinge bindingen worden getekend. De vierde sectie beschrijft wat er komt kijken bij de creatie van een add-on in Blender. In de laatste sectie wordt een conclusie getrokken over de onderwerpen die in dit hoofdstuk worden besproken.

3.1 Omvormen van het invoersbestand

3.1.1 Symmetrieoperaties in het CIF-formaat

Zoals in sectie 2.1.5 van deze tekst staat beschreven kan een kristal worden beschreven aan de hand van een aantal parameters en een lijst van elementen. Dit kan ook gezien worden in het CIF-bestand van Chaziet[Bijlage B] waar slechts vijf atomen worden beschreven terwijl het eenheidskristal in het totaal uit 150 atomen bestaat. Dit is erg handig aangezien er dertig keer zo weinig atomen moeten worden beschreven. In dit onderzoek vormt dit echter een probleem. Om een kristal te tekenen dient natuurlijk de positie van elk element gekend te zijn.

Aan de hand van de ruimtegroep en de centering van het kristalrooster kunnen alle symmetrieope-

raties worden verkregen. Deze symmetrieoperaties zijn in het CIF-formaat reeds gegeven in het blok `_symmetry_equiv_pos_as_xyz` als een lijst van fractionele coördinaten. Vanuit deze symmetrie-operaties kunnen de posities van elk element in het kristal berekend worden. Er dient rekening gehouden te worden met de mogelijkheid dat één bepaald element aan de hand van verschillende berekeningen kan verkregen worden zodat dit niet meermalen wordt genoteerd, en later getekend. Het is mogelijk een functie te schrijven die de lijst van symmetrieoperaties ophaalt en vervolgens voor elk van deze symmetrieoperaties de positie van de elementen berekenen die ze representeren. Er bestaan echter reeds programma's die dit soort berekeningen kunnen doen, OpenBabel is hier één van.

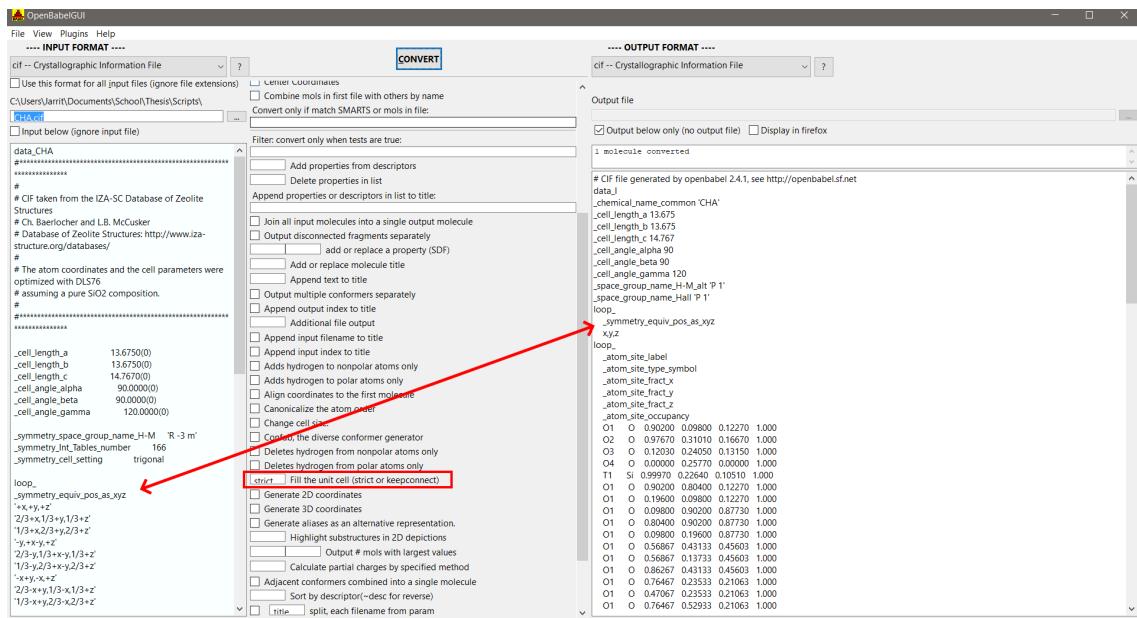
3.1.2 OpenBabel

OpenBabel is een open source chemische toolbox die gebruikt wordt om chemische data te zoeken, analyseren en te converteren.(OpenBabel, 2016) Naast het converteren tussen een groot aantal chemische dataformaten kan OpenBabel ook binnen eenzelfde formaat data converteren. Deze functionaliteit laat onder andere toe de kristaldata van een CIF-bestand om te zetten naar data van datzelfde kristal met een andere ruimtegroep. Door de ruimtegroep aan te passen naar dat van P 1, een vorm van P centrering waarbij er geen inwendige symmetrie bestaat, zal elk element dat beschreven wordt slechts op één plaats voorkomen in het kristal. In plaats van een lange lijst van symmetrieoperaties zal elk element dat gevormd kan worden door deze operaties, in het blok met de atoombeschrijvingen komen te staan.

De broncode van OpenBabel staat vrij beschikbaar op hun GitHub pagina. Hiernaast is het ook mogelijk de GUI-versie van OpenBabel te downloaden door de installatie-instructies te volgen die te vinden zijn op hun officiële webpagina.(OpenBabel, 2016)

In figuur[3.1] wordt de GUI van OpenBabel afgebeeld. In de linker- en rechterkolom van dit venster staan de in- en uitvoer van het programma en hun formaat. In de middelste kolom kunnen verschillende soorten van conversies worden aangeduid. In het geval van de afbeelding zijn zowel in- als uitvoer in het CIF-formaat en staat het *Fill the unit cel*-vakje op *strict*. Deze opties zorgen ervoor dat het invoerbestand zal worden omgezet naar een ander CIF-bestand maar ditmaal zonder symmetrieoperaties, wat duidelijk wordt bij het bekijken van de twee buitenste kolommen. Bij dit voorbeeld is er maar een optie aangeduid, er zijn nog een groot aantal erg handige keuzemogelijkheden maar deze komen voorlopig niet aan bod.

Hoewel de OpenBabel GUI het converteren van data erg eenvoudig en overzichtelijk maakt, zou het niet erg efficiënt zijn dat dit proces manueel moet worden gedaan voor elk kristal. Deze software kan echter ook vanuit een terminal worden uitgevoerd, wat, in combinatie met de subprocess module van python, de kristallografische interface toelaat deze conversie automatisch te laten verlopen. In hoofdstuk vijf wordt dit in meer detail uitgelegd.



Figuur 3.1: Conversie van ruimtegroep met OpenBabel GUI

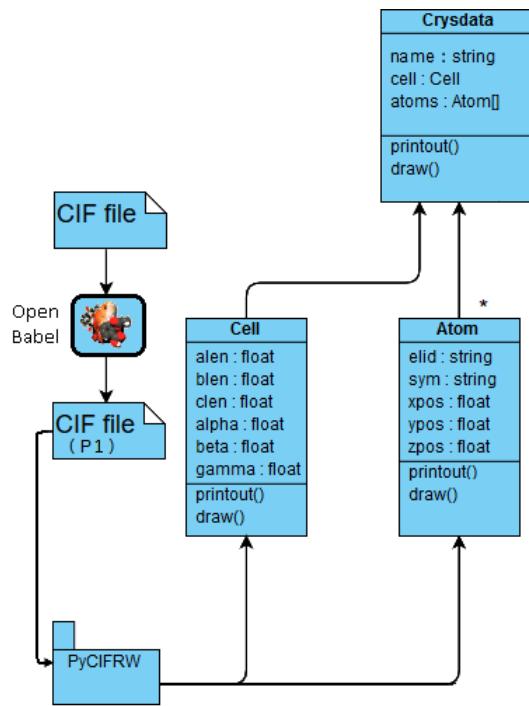
3.2 Van CIF naar py

3.2.1 Ontwerpen van datastructuren

Nu het CIF-bestand naar een meer bruikbaar formaat is omgezet, is er nood aan datastructuren waarin de informatie kan geparsed worden. Zo een datastructuur noemt men een klasse en het werken hiermee wordt objectgeoriënteerd programmeren genoemd. Enkele voordelen dat dit biedt over de klassieke, imperatieve methode van programmeren, zijn overzichtelijkheid, herbruikbaarheid van klassen en een extra laag van veiligheid doordat de data niet rechtstreeks kan worden aangepast.

Het inlezen van de data wordt in drie klassen gedaan. Een eerste klasse, *Cell*, die informatie bevat over het kristalrooster, waaronder de roosterparameters. Deze data-elementen worden ook wel de attributen van een klasse genoemd. Een tweede klasse, *Atom*, die de fractionele coördinaten van het atoom bevat en welk element het voorstelt. De derde klasse, *Crysdata*, overkoepelt voorgaande klassen in de zin dat deze naast de naam van het kristal ook één object van de klasse *Cell* en een lijst van objecten van de klasse *Atom* bevat. Hiernaast bezit elke klasse ook de methode, *printout()*, die de data van de klasse op een nette manier weergeeft op de terminal, en de methode *draw()* die later gebruikt wordt om het object te tekenen in Blender. Methodes zijn functies die op een object van een klasse kunnen worden opgeroepen. In Figuur[3.2] worden de onderlinge verhoudingen van deze klassen verduidelijkt, dergelijke figuur wordt een klassendiagramma genoemd, en geeft de algemene opbouw van een programma aan de hand van de gebruikte attributen, methodes, modules en hun onderlinge relaties. Uit dit diagram kan er geïnterpreteerd worden hoe de data uit het CIF-bestand met behulp van de PyCIFRW-parser zal worden ingelezen in de drie eerder

genoemde klassen.



Figuur 3.2: Vereenvoudigd klassendiagramma van het programma

3.2.2 Inlezen van CIF-bestanden

In de vierde sectie van hoofdstuk twee van deze tekst wordt het nut en de werking van een parser besproken. Er wordt ook gekeken naar twee bestaande programma's die in staat zijn CIF-bestanden te parsen en waarom PyCIFRW de voorkeur krijgt in dit onderzoek.

Door de module *CifFile* te importeren kunnen de schrijf- en leesfuncties van PyCIFRW worden opgeroepen in het programma. Nu kan het bestand, dat eerder werd omgezet met behulp van OpenBabel, worden geparsed. De geparsede data zal in de vorm van een soort dictionary ter beschikking zijn. Een dictionary is een python structuur die bestaat uit een lijst van sleutels, *keys*, en hun corresponderende waarden, *values* genoemd. De data kan verkregen worden door de dictionary op te roepen met een bepaalde sleutel.

De sleutels van de dictionary die de PyCIFRW parser aanmaakt stemmen overeen met de namen die te vinden zijn in het CIF-bestand zelf. Op deze manier kan de data dus bekomen worden die vervolgens in de klassen worden ingevuld.

3.3 Teken in Blender

3.3.1 Berekenen van de conversiematrix

Voor er kan getekend worden, dienen alle fractionele coördinaten omgezet te worden naar hun overeenkomstige orthogonale waarden. Dit wordt gedaan aan de hand van een functie die de conversiematrix berekend volgens de methode die beschreven wordt in de eerste sectie van hoofdstuk twee. Deze matrix wordt opgeroepen wanneer de conversie tussen fractioneel en orthogonaal moet gebeuren.

3.3.2 Omkadering van de eenheidscel

Het eerste en meest eenvoudige object dat getekend wordt zijn de randen van de eenheidscel. Deze acht ribben worden in feite reeds beschreven door de roosterparameters van het kristal. Het tekenen van deze wordt gedaan door de coördinaten van de hoekpunten te berekenen, en tussen elk hoekpunt en zijn drie 'buren' een ribbe te tekenen. Zo een ribbe wordt getekend als een cilinder aangezien lijnen geen volume bezitten en niet als vormen beschouwd worden in Blender.

Voor het berekenen van de hoekpunten is er niet echt nood aan de conversiematrix. Deze kunnen op een goniometrische manier berekend worden met behulp van de lengte van de ribben en hun onderlinge hoeken. Het is echter eenvoudiger de fractionele waarden van de hoekpunten te gebruiken aangezien elke coördinaat van zo een hoekpunt steeds een nul of een één moet zijn. Door de fractionele coördinaten met de conversiematrix om te zetten, kunnen de orthogonale coördinaten van de hoekpunten verkregen worden. Ten slotte kunnen de cilinders getekend worden tussen deze punten. Door alle cilinders die de omkadering vormen, samen te nemen als één object wordt de uiteindelijke omkadering bekomen.

3.3.3 Atomen

Vervolgens dient de eenheidscel opgevuld te worden met de atomen van het kristal. De lijst van alle elementen en hun positie binnen het kristal is te vinden als een attribuut van de klasse *Crys-data*. Deze lijst wordt doorlopen en op elk element zal zijn methode *draw()* worden opgeroepen. Deze methode zal met behulp van de conversiematrix de orthogonale coördinaten van het atoom berekenen. Met behulp van een dictionary die voor elk element de straal van het atoom bevat, kan vervolgens een bol getekend worden met de correcte afmetingen. En ten slotte, aan de hand van een andere dictionary die de kleur van de elementen bevat, krijgt deze bol een kleur toegekend.

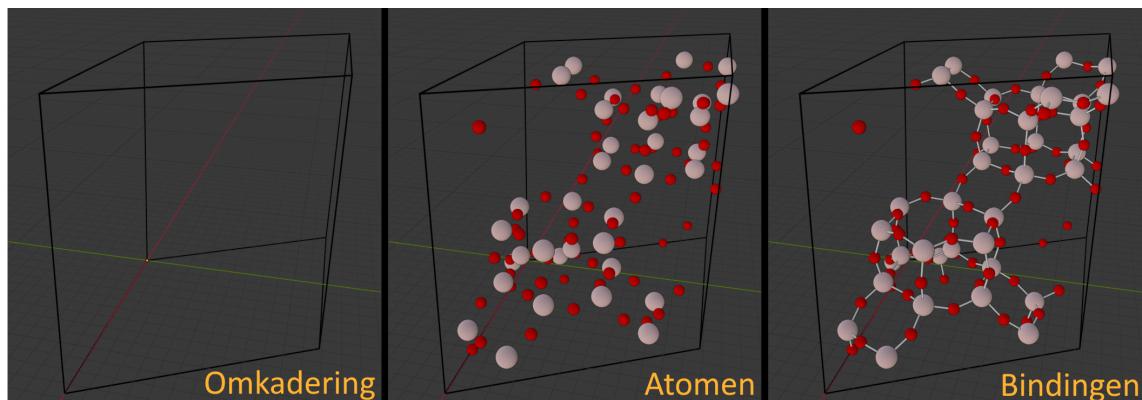
3.3.4 Atoombindingen

Het programma laat ook toe bindingen tussen atomen te tekenen, op basis van hun onderlinge afstand. De gebruiker kan een variabele aanpassen die bepaalt hoe groot de afstand tussen twee atomen maximaal mag zijn, zodat de binding tussen deze gemaakt zal worden.

Het programma zal vervolgens weer de lijst met atomen doorlopen en voor elk atoom de afstand tussen zichzelf en alle andere berekenen. In het geval dat deze onderlinge afstand tussen de twee atomen kleiner is dan de door de gebruiker gekozen waarde, zal er een binding aangemaakt worden. Het tekenen van een binding zou op een gelijkaardige manier kunnen gedaan worden als het maken van de omkadering, er wordt echter een andere manier gebruikt die de bindingen zal vasthechten aan het atoom. Dankzij deze feature zal, wanneer de gebruiker een atoom versleept, het uiteinde van de binding vast blijven hangen aan het atoom.

Het programma zal tussen de twee bollen die de atomen voorstellen een *bézierkromme* tekenen, en aan de middelpunten van deze bollen een uiteinde van deze kromme vasthechten. Ten slotte zal er een *bevel* van een cirkel worden gedaan over deze kromme. Zo'n *bevel* kan worden gezien als een manier van tekenen waarbij de kromme een soort gids is waarover een cirkel wordt geëxtrudeerd. Zo wordt rond de kromme een cilindervormig figuur gevormd die vasthangt aan beide atomen.

Op Figuur[3.3] zijn de drie stappen van het tekenen van een eenheidskristal te zien, eerst de omkadering, dan de atomen en ten slotte de bindingen tussen de atomen.



Figuur 3.3: De drie stappen in het tekenen van een kristal

3.4 Creëren van een Blender add-on

3.4.1 Blender add-ons

De laatste stap in het ontwerpproces is het aanmaken van een interface tussen de gebruiker en het programma, zodat de gebruiker op een eenvoudige manier zijn CIF-bestanden kan visualiseren. Blender laat toe zelf add-ons te creëren. Zo een add-on is een programma dat in de Blender interface kan worden opgeroepen door op de F3 toets te drukken, het is zelfs mogelijk een add-on een eigen venster te geven of in een reeds bestaand venster in te voegen.

Het schrijven van een add-on kan worden gedaan in de Scripting omgeving van Blender, waar het meteen kan uitgevoerd worden of in een externe IDE of tekstbewerkingsprogramma zodat de add-on later kan worden geïnstalleerd. Een add-on hoeft slechts een keer geïnstalleerd te worden en

kan vanaf dan worden herladen door de *Reload Scripts* functionaliteit uit te voeren in Blender. Op deze wijze kan een externe IDE of editor gebruikt worden, welke veel handiger in omgang is dan de text editor van Blender zelf, terwijl de add-on constant kan getest worden na elke aanpassing van het script.

3.4.2 Opbouw van een add-on script

Een Blender add-on moet steeds een bestand bevatten met de naam `__init__.py`, dit is het bestand dat Blender eerst zal uitvoeren wanneer de add-on wordt aangemaakt. In dit bestand staat steeds een header gedeelte dat `bl_info` heet. Deze variabele is een dictionary waarin de details van de add-on worden vermeld, waaronder de naam van de add-on, de versie van Blender waarvoor deze is gemaakt, de naam van de auteur en in welke editor omgeving deze add-on kan worden gebruikt. Een header is nog niet genoeg om een werkende add-on te hebben, bij het uitvoeren van een add-on gaat Blender zoeken naar een *register* methode. Deze *register* methode zal telkens worden opgeroepen wanneer Blender de add-on inlaadt en zullen alle onderdelen van de add-on worden ingeladen. Deze onderdelen worden in de volgende secties verder beschreven. Wanneer de add-on wordt uitgeschakeld zal de *unregister* methode worden opgeroepen welke het tegenovergestelde doet van de eerder beschreven *register* methode.

3.4.3 Operatoren

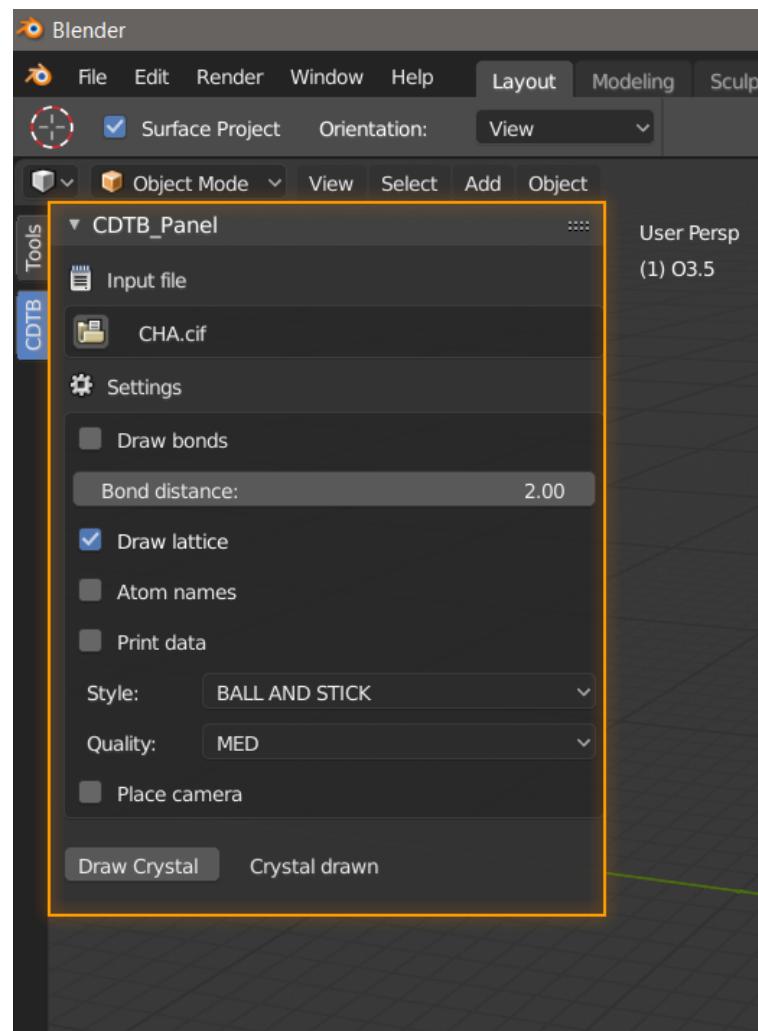
Operatoren, of *operators*, zijn klassen in een add-on waarin de uitvoerbare elementen worden beschreven. Deze bevatten telkens hun eigen *register* en *unregister* methodes. In de *register* methode worden de attributen van de operator beschreven, welke Blender zal aanmaken bij het inladen van de add-on. Deze attributen kunnen dan eventueel uit het geheugen worden verwijderd bij het uitvoeren van de *unregister* methode wanneer de add-on wordt uitgeschakeld.

Een operator kan ook een *execute* methode bevatten, deze methode zal worden opgeroepen wanneer deze operator in de add-on wordt uitgevoerd. In deze methode kunnen de attributen worden aangepast of ingelezen, kunnen andere functies worden opgeroepen of kan code staan die moet worden uitgevoerd zodra de operator wordt opgeroepen.

In het programma worden twee operators aangemaakt. Eén operator heeft als functie een bestandsbrowser te openen waarmee de gebruiker het CIF-bestand kan selecteren dat moet worden gevisualiseerd. De tweede operator voert het hele script uit dat in dit hoofdstuk beschreven wordt, vanaf het converteren van het bestand met OpenBabel tot het tekenen van het kristal in Blender. Deze operatorklasse heeft een groot aantal attributen die de gebruiker kan aanpassen, waaronder enkele *booleans* die kunnen aan of uitgezet worden, een *float* die de maximale bindingdsafstand van de atomen bepaalt en twee *enumerations* die de tekenwijze en de kwaliteit van het tekenen bepalen. Een enumeratie is een datastructuur die een verzameling is van symbolische namen, of leden, die gebonden zijn aan unieke, constante waarden. Binnen een enumeratie kunnen de leden met elkaar worden vergeleken en kan er over de leden worden geïtereerd.(Mike, 2018) Het nut van deze enumeraties komt in de volgende sectie aan bod.

3.4.4 Panelen

Panelen, of *panels* zijn klassen die als voornaamste functie het visualiseren van de add-on op het Blender scherm hebben. Een paneelklasse heeft enkele attributen waaronder de naam van het paneel en in welke omgeving en venster de add-on wordt weergegeven. Ook deze klasse bevat een *register* en *unregister* methode, om het paneel te initialiseren. Hiernaast is er nog de *draw* methode, waarin de lay-out van het paneel kan worden aangepast. Hiermee kunnen de attributen van de operatoren worden weergegeven zodat ze door de gebruiker kunnen worden aangepast. Boolean worden weergegeven als selectievakjes, getallen als schuifbalken en tekstvakjes en de eerder vernoemde enumeraties als een keuzelijst van de elementen die het bevat. Hiernaast kunnen er labels gemaakt worden waarin tekst kan worden afgebeeld en worden de operators als een drukknop weergegeven die hun *execute* methode uitvoert wanneer de gebruiker op deze knop klikt. De opmaak van het paneel kan worden aangepast door middel van kleuren, pictogrammen, het invoegen van scheidingen, het gebruik van rijen en kolommen en door onderdelen van het paneel in aparte kaders onder te brengen.



Figuur 3.4: Het paneel van de add-on in de Blender GUI

Het paneel van de add-on van het programma, Figuur[3.4], is terug te vinden in de *3D view* omgeving van Blender onder het *Tools* venster. Hiernaast is de add-on enkel zichtbaar wanneer de gebruiker zich in *Object mode* bevindt. Het paneel zelf wordt opgedeeld in twee kaders. In de eerste kader staat een drukknop, die de bestandsbrowseroperator uitvoert, met een mapje als pictogram. Naast deze knop staat een label die het pad naar het geselecteerde bestand laat zien. In de tweede kader zijn alle instellingen zichtbaar die invloed hebben op het tekenen van het kristal. Hier zijn onder andere de twee keuzelijsten weergegeven. Het veld onder de kaders is opgedeeld in twee kolommen, in de eerste kolom staat de drukknop *Draw Crystal*, die de tweede operator zal uitvoeren waarmee het kristal zal getekend worden. Rechts van deze knop staat nog een label die eventuele opmerkingen aan de gebruiker zal geven, bijvoorbeeld wanneer het kristal is getekend of wanneer er op de knop wordt geklikt zonder dat er een bestand is geselecteerd.

3.5 Conclusie

In dit hoofdstuk werden het ontwerpproces en de werking van het programma op een beknopte manier beschreven. Er werd kennismaking gemaakt met OpenBabel en hoe deze in het programma wordt gebruikt om de symmetrie binnen een kristal om te zetten naar een beter leesbaar formaat. Er is dieper ingegaan over hoe PyCIFRW de kristaldata uit CIF-files beschikbaar stelt in de vorm van een dictionary en hoe deze verder in worden gevuld in de geschikte klasse.

Om het kristal correct te visualiseren is er nood aan een conversiematrix die de fractionele coördinaten kan omzetten naar orthogonale. Met behulp van deze matrix zal eerst een omkadering van het eenheidskristal getekend worden. Dit wordt gedaan door cilinders te tekenen tussen de hoekpunten van de cel. Vervolgens worden de atomen een voor een getekend als gekleurde bollen met verschillende afmetingen op basis van het element. Ten slotte worden de afstanden tussen de atomen berekend zodat er al dan niet een binding kan worden getekend.

Het maken van een add-on in Blender laat de gebruiker toe het script op een eenvoudige manier uit te voeren, al dan niet met behulp van een grafische tool. Een add-on bestaat uit een header met informatie over de add-on, één of meerdere operatorklassen die attributen kunnen bezitten en een bepaalde functie uitvoeren wanneer ze worden opgeroepen en een paneelklasse waarmee het visuele aspect van de add-on kan worden gepersonaliseerd. Op zo een paneel kunnen de attributen door de gebruiker worden aangepast en kunnen de operatoren worden uitgevoerd via drukknoppen.

Hoofdstuk 4

Gedetailleerd ontwerpproces

In hoofdstuk drie wordt een algemeen overzicht gegeven van de opbouw en werking van het programma. Er wordt echter niet dieper ingegaan op de onderliggende code van het programma. Dit hoofdstuk zal zich voornamelijk verdiepen in deze code, de structuur ervan en zullen de belangrijkste onderdelen ervan worden uitgelegd. Hiernaast zal er ook worden beschreven hoe de externe programma's moeten worden geïnstalleerd en hoe deze in het programma worden geïmplementeerd.

Om de gedachtengang achter het ontwerpproces makkelijker volgbaar te maken zal er in dit hoofdstuk vooral gebruikt gemaakt worden van een actieve schrijfwijze en de wetenschappelijke 'we'-vorm. De code zal worden uitgelegd aan de hand van listings die rechtstreeks uit de code van het programma komen. De volledige code kan worden teruggevonden in de bijlagen van deze tekst.[Bijlage D] Het is mogelijk dat de lijnnummers die in de tekst voorkomen niet volledig overeenstemmen met de lijnnummers van het gehele bestand. Dit is te wijten aan het feit dat de code kan zijn aangepast na het afdrukken van deze tekst.

In de eerste sectie zal de installatie van de externe programma's worden besproken. Eerst zal er gezien worden hoe de OpenBabelGUI moet geïnstalleerd worden. Hierna zal worden uitgelegd hoe PyCIFRW kan worden verkregen. De tweede sectie beschrijft aanvankelijk hoe er kan gecontroleerd worden of een bestand een CIF-bestand is. Vervolgens wordt er uitgelegd hoe OpenBabel en de CifFile module kunnen worden gebruikt in een Python script. Het tekenen van de celomkadering, atomen en bindingen met behulp van de bpy module wordt uitgelegd in de derde sectie van dit hoofdstuk. In de voorlaatste sectie wordt er verteld hoe een Blender add-on wordt opgebouwd en hoe deze ontworpen wordt. In de conclusie van dit hoofdstuk worden de belangrijkste punten van dit hoofdstuk nogmaals op een rijtje gezet.

4.1 Installatie van externe programma's

Zoals in hoofdstuk drie wordt beschreven gaan we gebruik maken van twee externe programma's die we zullen nodig zullen hebben om het programma uit te kunnen voeren, OpenBabel en PyCIFRW. Deze sectie legt uit hoe we deze kunnen installeren op een systeem. Hoewel deze in

het geval van dit onderzoek met Windows 10 wordt gewerkt, werken deze software ook op oudere versies van Windows. Ons programma is gericht op besturingssysteemonafhankelijkheid, dit wil zeggen dat het ook op Linux en Apple moet werken. Het is echter mogelijk dat het installeren van deze externe programma's anders verloopt dan op Windows. De installatie van deze programma's zal in deze tekst echter enkel voor Windowsystemen worden uitgelegd.

4.1.1 OpenBabel

De installatie van OpenBabel kan gevonden worden op hun webpagina: <http://openbabel.org/wiki/Category:Installation> en is gratis voor iedereen. Op deze pagina gaan we de OpenBabelGUI *installer* downloaden, zie Figuur[4.1], let op dat de bit-versie met die van ons systeem overeenkomt. De download zou automatisch moeten starten. Ten slotte kunnen we de installatie-wizard starten door het gedownloade bestand uit te voeren.

Na het volgen van de installatiewizard zou de OpenBabel GUI moeten geïnstalleerd zijn op ons systeem, en kan het gebruikt worden in ons programma. Dit zal in het verdere verloop van dit hoofdstuk worden uitgelegd.

4.1.2 PyCIFRW

Om de PyCIFRW module op ons systeem te installeren hebben we nood aan een prompt waarop Python3.7 kan worden uitgevoerd. Zo'n prompt kan verkregen worden door Anaconda te installeren en gebruik te maken van *Anaconda prompt*. Vervolgens hebben we de Python pip installer nodig. Deze is automatisch aanwezig op Python3.7

Pip laat ons toe allerlei Python modules te installeren, waaronder de PyCIFRW module. Dit doen we met door volgend commando in te geven in het prompt:

```
pip install PyCifRW
```

Dit zorgt ervoor dat we nu toegang hebben tot de PyCIFRW module wanneer we Python uitvoeren en kan eenvoudig getest worden door Python op te starten en de volgende lijn code uit te voeren:

```
import CifFile
```

Als er geen foutmelding wordt gegeven, wilt het zeggen dat de module correct is geïnstalleerd op onze Python installatie.

Doordat Blender een eigen Python installatie heeft, zal deze nog geen toegang hebben tot onze geïnstalleerde modules. Dit kunnen we oplossen door de map die de PyCIFRW module bevat te kopiëren naar de Python installatie van Blender. Deze map kunnen we vinden op plaats waar we Python hebben geïnstalleerd op ons systeem. In het geval dat Anaconda wordt gebruikt, vinden we deze module in de submap *site-packages* van de installatiemap van Anaconda.(*Anaconda3 >> Lib >> site-packages*). Vervolgens moeten we deze map *CifFile* kopiëren naar de Python installatie van Blender (*blender-2.80.0-git.3c8c1841d72-windows64 >> 2.80 >> python >> lib >> site-packages*).

Een alternatieve methode om de PyCIFRW module werkende te krijgen op Blender, zonder nood te hebben aan een Python3.7 installatie, is door de *CifFile* map rechtstreeks in de Pythoninstallatiemap van Blender te plaatsen. De *CifFile* kan gedownload worden vanaf de GitHub pagina van deze thesis: <https://github.com/JarritB/Thesis>

4.2 Inlezen van het bestand

4.2.1 Controleren van het bestand

Vooraleer we het bestand kunnen omzetten met OpenBabel moeten we controleren of het gekozen bestand wel degelijk een CIF-bestand is, dit zou anders voor problemen kunnen zorgen in het verdere verloop van het programma.

Om dit te controleren moeten we nakijken of ons bestand de extensie *.cif* heeft, in sommige gevallen wordt de extensie met hoofdletters geschreven, hiermee dienen we dus ook rekening te houden. Omdat de bestandsnaam het volledige pad naar het bestand inhoudt, moeten we enkel de vier laatste tekens overhouden. Dit wordt gedaan op lijn 730 van Listing[4.1].

De variabele *ext* zal nu enkel de laatste vier tekens van de filenaam bevatten. Bij een CIF-bestand zullen deze laatste vier altijd *.cif* zijn, we moeten de variabele dus hiermee vergelijken. Door de *.lower()* methode op te roepen op de extensie zal deze altijd naar kleine letters worden omgezet, en hoeven we dus geen rekening te houden met hoofdletters. In het geval dat de extensie niet gelijk is aan *.cif* gaan we een foutbericht weergeven in de terminal en gaan we in de *user_feedback* variabele ook een bericht zetten die zal verschijnen op onze add-on, deze variabele wordt later verder uitgelegd. Omdat er een fout bestand is ingegeven zal het programma beëindigd worden met een *return*, het heeft geen zin dit bestand proberen te tekenen. In listing[4.1] wordt de controle van het bestand gedaan.

```
ext = file[len(file)-4:]
if(ext.lower() != ".cif"):
    print("Only cif files can be visualised")
    user_feedback = "Not a cif file"
    return
```

Listing 4.1: Controle van de extensie

4.2.2 Uitvoeren van OpenBabel

Nu we zeker weten dat we met een CIF-bestand te maken hebben kunnen we de inwendige symmetrie van het kristal wegwerken met OpenBabel.

Het oproepen van OpenBabel doen we met behulp van de *subprocess* module van python. Wanneer we deze module oproepen zal er een nieuw process worden gestart, die in ons geval OpenBabel zal uitvoeren.

Voor we de *subprocess* module kunnen oproepen in ons script moeten we deze importeren met de lijn code in Listing[4.2]

```
import subprocess
```

13

Listing 4.2: Importeren van de subprocess module

Vooraleer we OpenBabel gaan oproepen gaan we controleren of OpenBabel wel geïnstalleerd is op ons systeem. Dit wordt gedaan met een *try-except* blok, zie Listing[4.3], welke gaat proberen de code in het *try* gedeelte uit te voeren. Als dit niet lukt zal het programma in plaats daarvan het *except* gedeelte uitvoeren. Bij een correcte installatie van OpenBabel zal de *obabel_fill_unit_cell* functie worden uitgevoerd, zie lijn 739 van Listing[4.3].

In het geval dat OpenBabel niet kan worden opgeroepen gaan we een foutbericht weergeven in de terminal en in de *user_feedback* variabele en zal ons programma verder werken met het ingegeven CIF-bestand. Hoewel het kristal niet kan geconverteerd worden, zullen we toch een deel van het kristal kunnen tekenen. In sommige gevallen heeft het ingegeven kristal geen inwendige symmetrie, en is er geen nood aan OpenBabel om het kristal te kunnen visualiseren.

```
try : 736
    # Convert the cif file to its P1 symmetry notation as a 737
    temporary cif file
    print('Converting %s to P1' %file) 738
    obabel_fill_unit_cell(file , "temp.CIF") 739
    cf = CifFile("temp.CIF") 740
except : 741
    print("No OpenBabel installation found, install it from http 742
          ://openbabel.org/wiki/Category:Installation")
    user_feedback = "OpenBabel not installed" 743
    cf = CifFile(file) 744
```

Listing 4.3: Controle van de OpenBabel installatie

De functie *obabel_fill_unit_cell* heeft het pad naar het ingevoerde CIF-bestand en de naam van het geconverteerde CIF-bestand als argumenten en bestaat uit slechts één lijn code. Op deze lijn roepen we de *run* methode op de *subprocess* module op, deze heeft als argument een *string* waarin het uit te voeren commando staat zoals het in een prompt zou worden uitgevoerd. Het commando waarmee we OpenBabel uitvoeren, bestaat uit volgende parameters:

- obabel: roept OpenBabel op
- -icif: type van het invoerbestand, wordt gevolgd door de bestandsnaam
- -ocif: type van het uitvoerbestand
- -O: gevolgd door naam van het uitvoerbestand
- –fillUC: selecteert de mode waarin de symmetrie zal worden omgezet

- keepconnect: parameter van de fillUC mode, tekent ook atomen buiten de eenheidscel

In onze code ziet het er dan als volgt uit:

```
subprocess.run(['obabel', '-icif', cif_file, '-ocif', '-O',
               p1_file, '--fillUC', 'keepconnect'])
```

675

Listing 4.4: Uitvoeren van OpenBabel

4.2.3 Zelf een parser ontwerpen

In de vierde sectie van hoofdstuk twee werd er reeds aangehaald wat er allemaal komt kijken bij het ontwerpen van een parser. Het is gelukt een programma[Bijlage C] te ontwerpen dat alle CIF-bestanden kan inlezen die te vinden zijn op de kristallografische databank van het IZA.(IZA, 2018) Het parsen van deze bestanden lukte echter enkel omdat deze allemaal dezelfde tekststructuur bezitten. Bestanden uit databanken die werken met een andere structuur kunnen mogelijk verkeerd worden geïnterpreteerd waardoor de parser zijn nut verliest. Het is mogelijk de code van de parser aan te passen zodat, ongeacht de structuur van het CIF-bestand, een juiste interpretatie van de data kan worden gedaan. Dit vergt echter veel tijd en zal, gezien er reeds werkende CIF-parsers bestaan, niet verder worden onderzocht.

4.2.4 Parsen met PyCifRW

Het geconverteerde bestand parsen gaan we doen met behulp van de *CifFile* module van PyCifRW. Om toegang te krijgen tot deze module moeten we deze eerst importeren in onze code, zie Listing[4.5]. Omdat er hier een fout wordt gegeven als de module niet geïnstalleerd is, moeten we deze import omsluiten met een *try-except* blok. Als de module niet kan geïmporteerd worden gaan we een variabele aanzetten zodat het programma later weet dat het een foutmelding moet geven aan de gebruiker.

```
try :  
    from CifFile import CifFile  
    pars_check = False  
except:  
    print("PyCifRW not installed , try: pip install PyCifRW")  
    pars_check = True
```

16
17
18
19
20
21

Listing 4.5: Importeren van de *CifFile* module

Met de *CifFile* module kunnen we vervolgens een *CifFile* object aanmaken. Door de bestandsnaam mee te geven als argument zal de *CifFile* module automatisch het bestand parsen en de data opslaan in het object, dit wordt gedaan op lijnen 740 en 744 van Listing[4.3]. Naast het *CifFile* object maken we ook een object van de *Crysdata* klasse aan. Deze klasse heeft een initialisatiemethode die zal worden uitgevoerd zodra een object ervan wordt aangemaakt, zie Listing[4.6]. In deze

methode gaan we de attributen van de klasse bepalen. Enkele interessante attributen van deze klasse zijn:

- start: start een soort van timer op, zo kan de loopduur van het programma worden nagekeken
- cell: een object van de *Cell* klasse, bevat de roosterparameters
- atoms: een lijst van objecten van de klasse *Atom*, alle atomen van het eenheidskristal
- pos: een lijst met alle inwendige symmetrieën van het kristal, deze wordt voorlopig niet gebruikt
- ftoc: de fractionele naar orthogonale conversiematrix, deze wordt berekend aan de hand van de roosterparameters

In Listing[2.7] van hoofstuk twee werden de voornaamste functies van de *CifFile* module reeds beschreven. In hoofdstuk drie zagen we ook dat zo een *CifFile* object kan gezien worden als een dictionary. Alleenstaande gegevens in het datablok van het CIF-bestand, zoals de roosterparameters, kunnen we eenvoudig opvragen met de correcte sleutels. Dit doen we om een object van de klasse *Cell* aan te maken, zie Listing[4.6].

```
def __init__(self, cb): 576
                      577
    self.alen = float(cb["_cell_length_a"]) 578
    self.blen = float(cb["_cell_length_b"]) 579
    self.clen = float(cb["_cell_length_c"]) 580
    self.alpha = float(cb["_cell_angle_alpha"]) 581
    self.beta = float(cb["_cell_angle_beta"]) 582
    self.gamma = float(cb["_cell_angle_gamma"]) 583
```

Listing 4.6: Initialisatiemethode van de klasse Cell

Het toekennen van het object *atoms* doen we met een aparte functie. In deze functie gaan we nieuwe *Atom* objecten aanmaken, deze invullen met data door het lesblok met atom in *CifFile* object te doorlopen en deze in een lijst plaatsen. Dit wordt gedaan in Listing[4.7]

```
def readEl(cb): 639
                  640
    elements = [] 641
    previd = [] 642
    idcnt = [] 643
    lb = cb.GetLoop("_atom_site_label") 644
    for el in lb: 645
        flag = False 646
        for i in range(len(previd)): 647
            if (el[0] == previd[i]): 648
```

```

        flag = True           649
        break                650
if(flag):
    idcnt[i] += 1       651
else:
    previd.append(el[0]) 652
    idcnt.append(0)      653
    i = len(idcnt)-1   654
    id_t = "{}.{}".format(el[0], idcnt[i]) 655
    elements.append(Atom(id_t, el[1], el[2], el[3], el[4])) 656
return elements          657

```

Listing 4.7: Functie die een lijst van atomen aanmaakt

In de conversie die OpenBabel doet krijgen de atomen dezelfde ID toegekend als het atoom waarvan de symmetriepositie is berekend, om dit op te lossen dienen we een lijst bij te houden met de ID's van de atomen die we al hebben aangemaakt en een teller bij te houden van het aantal van atomen met deze ID. Als er een atoom met hetzelfde ID voorkomt, zal er aan dit ID een nummer worden toegevoegd en gaan we de teller met één verhogen. Op deze manier zijn we zeker dat elk atoom een eigen ID heeft waarmee het later kan worden aangesproken.

4.3 Tekenen in Blender

4.3.1 Programmeren met de Blender API

Tot nu toe zijn we in staat een CIF-bestand te converteren met OpenBabel, vervolgens te parsen met PyCIFRW en een *Crysdata* object te creëren waarin de kristalinformatie staat. De volgende stap is het visualiseren van deze data in de Blender omgeving. Hiervoor gaan we gebruikmaken van de Blender API en meer specifiek, de bpy module die de Blender API aanbiedt.

Zoals met elke module die we tot nu toe gebruikt hebben, moeten we de bpy module eerst importeren. Omdat deze module niet bereikbaar is buiten de Blender omgeving gaan we weer gebruikmaken van een *try-except* blok, welke zal proberen de module te importeren. Als dit lukt gaan we een variabele aanzetten zodat ons programma weet dat het mogelijk is deze module te gebruiken, zoniet zal het een foutbericht geven, maar zal alles anders wel worden gedaan zodat het programma kan getest worden buiten de Blender omgeving.

Het tekenen van het kristal is een methode van de *Crysdata* klasse, deze zal stap voor stap functies aanroepen waarin bepaalde onderdelen van het kristal worden getekend. Wanneer de gebruiker zich niet in de Blender omgeving bevindt zal ons programma deze methode niet oproepen.

4.3.2 Tekenen van de omkadering

Het tekenen van de omkadering van de eenheidscel is het eerste wat er zal gebeuren. Omdat het tekenen van de omkadering een optie is die de gebruiker kan aanzetten, gaan we eerst controleren of deze stap wel moet gebeuren, anders wordt deze overgeslagen.

De *drawCell* methode is er een van de *Crysdata* zelf. Deze methode zal met behulp van de conversiematrix en de roosterparameters, welke te vinden zijn als attributen van het *cell* object, eerst kleine bollen tekenen op de hoekpunten van de eenheidscel. Dit wordt gedaan in Listing[4.8].

```

for i in range(2): 451
    for j in range(2): 452
        for k in range(2): 453
            bpy.ops.mesh.primitive_uv_sphere_add(size=
                lattice_size, location=toCarth(self.ftoc,[i,j,k
                ]))
            activeObject = bpy.context.active_object # Set 455
            active object to variable
            cell_corners.append(activeObject) 456
            mat = bpy.data.materials.new(name="MaterialName") 457
            # set new material to variable
            activeObject.data.materials.append(mat) # add the 458
            material to the object
            bpy.context.object.active_material.diffuse_color 459
            = [0,0,0] # change color

```

Listing 4.8: Teken en kleuren van de hoekpunten van de eenheidscel

Dit is het ideale moment om enkele, in ons programma vaak voorkomende, functies van de *bpy* module uit te leggen. Het aanmaken van nieuwe vormen wordt gedaan met de *bpy.ops.mesh* functie. Op lijn 454 van Listing[4.8] roepen we deze functie op om een *primitive_uv_sphere* te tekenen. Deze vorm is een type van bol. In de volgende lijnen gaan we onze net aangemaakte bol als actief object zetten zodat we aan dit object een nieuw materiaal kunnen toekennen. Door een vorm een materiaal toe te kennen kunnen we een kleur geven aan het materiaal, en aan onze bol. Kleuren worden in Blender aan de hand van RBG-waarden bepaalt, deze waarden worden in de vorm van een lijst gegeven waarin het niveau van elke kleur (rood, groen, blauw) met een getal tussen nul en één wordt voorgesteld. In het geval van onze bol kiezen gebruiken we de waarden [0,0,0], welke zwart voorstellen.

Nu we onze hoekpunten hebben bepaald, en getekend, moeten we de ribben van onze omkadering tekenen. Dit gaan we doen door cilinders te tekenen tussen de hoekpunten van de eenheidscel. Door een lijst bij te houden waarin de bollen worden opgeslagen wanneer ze worden aangemaakt, zie lijn 456 van Listing[4.8], kunnen de hoekpunten worden verkregen. Omdat we niet tussen alle hoekpunten een lijn willen tekenen, dit zou namelijk ook de diagonalen tekenen, gaan we specificiëren tussen dewelke we een lijn willen tekenen. Door de hoekpunten in een lijst bij te houden

zijn deze in essentie genummerd. We gaan twee lijsten aanmaken die elk één nummer bevatten van de koppels van hoekpunten die we willen verbinden. Door gebruik te maken van de *zip* functie van Python kunnen we deze lijsten samennemen en er twee variabelen tegelijk over itereren. Deze twee variabelen stellen telkens het paar van hoekpunten voor waartussen we een cilinder gaan tekenen. Vervolgens gaan we aan de hand van de locaties van de hoekpunten de afstand ertussen berekenen, dit wordt de lengte van onze cilinder. Het tekenen van de cilinder wordt op een gelijkaardige manier gedaan als we eerder hebben gedaan bij het tekenen van de hoekpunten. De cilinder zal vertrekken op de plaats van het eerste hoekpunt en met behulp van enkele goniometrische berekeningen kunnen we de cilinder zo roteren dat het uiteinde overeenkomt met het andere hoekpunt.

```

for i in cell_corners:
    i . select_set(action="SELECT")                                464
for i in cell_edges:
    i . select_set(action="SELECT")                                465
# set corner in origin as active and join meshes as one      466
object
bpy.context.view_layer.objects.active = cell_corners[0]          467
bpy.ops.object.join()                                         468

```

Listing 4.9: Samennemen van alle objecten in een vorm

Ook de ribben van de cel gaan we bijhouden in een lijst. Op deze manier kunnen we na het tekenen van de ribben alle objecten uit de lijsten van hoekpunten en ribben selecteren en samenvoegen in één object. Dit wordt gedaan in Listing[4.9].

4.3.3 Teken van atomen

Het tekenen van de atomen doen we met de *drawAtoms* methode van de *Crysdata* klasse. Deze methode zal de lijst met atomen aflopen en op elk element van deze lijst de *drawObj* methode oproepen. De *drawObj* methode is een methode van de *Atom* klasse en wordt weergegeven in Listing[4.10]

```

def drawObj(self ,ftoc):                                         609
    size = sizedic [self .sym]* styledic [draw_style ][0]+ bond_radius* 610
    styledic [draw_style ][1]
    bpy.ops.mesh.primitive_uv_sphere_add(segments=qualitydic [ 611
        draw_quality ], ring_count=qualitydic [ draw_quality ]/2 ,size=
        size ,location=toCarth(ftoc ,[ self .xpos ,self .ypos ,self .zpos
        ]))
    bpy.context.object.name = self .elid                           612
    activeObject = bpy.context.active_object # Set active object 613
    to variable

```

```

mat = bpy.data.materials.new(name="MaterialName") # set new      614
      material to variable
activeObject.data.materials.append(mat) # add the material to 615
      the object
if (atom.name):                                         616
    bpy.context.object.show_name = True                617
if (atom_color):
    bpy.context.object.active_material.diffuse_color = 619
        colordic[self.sym] # change color to dictionary color
else:
    bpy.context.object.active_material.diffuse_color = 620
        [1,1,1] # change color to white

```

Listing 4.10: De tekenmethode van de klasse Atom

Om elk element een eigen grootte en kleur te geven maken we gebruik van twee dictionaries, het is mogelijk deze samen te nemen in één dictionary om minder geheugen te gebruiken, dit maakt het echter minder overzichtelijk voor de gebruiker. Omdat deze dictionaries meer dan 90 elementen bevatten, worden deze niet rechtstreeks in de code opgenomen, maar zullen we ze in een extern bestand opslaan. Op deze manier hoeft een gebruiker het bronbestand van het programma niet aan te passen, wanneer deze de dictionaries wil bewerken. Deze externe dictionaries kunnen worden gevonden als bijlagen van deze tekst.[Bijlage E & F] Het inlezen van een dictionary vanuit een extern bestand wordt gedaan met behulp van de *eval* functie. Deze functie heeft als argument een string, welke *eval* als Python code zal interpreteren. Het inlezen van een externe dictionary wordt gedaan in Listing[4.11].

```

with open(path+dir_sep+'colordic.txt','r') as inf:          88
    colordic = eval(inf.read())

```

Listing 4.11: Inlezen van een dictionary vanuit een extern bestand

De tekenmethode van het atoom zal, met het symbool van het element als sleutel, de straal van het atoom uit de *sizedic* dictionary halen. Om de positie van het atoom te bepalen, wordt de conversiematrix gebruikt. Deze zal de fractionele coördinaten, die attributen zijn van het *Atom* object, omzetten naar hun orthogonale waarden. Dan tekenen we een bol met de eerder bepaalde straal op die plaats. De straal hangt ook af van de stijl waarin het kristal zal worden getekend. Bij de *STICK AND BALL* stijl zal de straal worden gehalveerd, zodat de bindingen zichtbaar worden. Bij de *STICK* tekenstijl zal de straal zo klein zijn dat enkel de bindingen nog zichtbaar zijn. De argumenten *segments* en *ring_count* op lijn 611 van Listing[4.10] bepalen uit hoeveel vlakken de bol bestaat, bij een lage waarde zal de bol eerder hoekig zijn. De gebruiker kan het aantal segmenten kiezen waarmee het aan de hand van een dictionary wordt ingevuld.

Op lijn 612 van Listing[4.10] kennen we de getekende bol een naam toe, via deze naam kunnen we onze bol later terugvinden in de lijst van getekende objecten. Hierna gaan we onze bol een materiaal en een kleur toekennen. De kleur wordt bepaald door de waarde die terug te vinden is in

colordic, dit is de dictionary waarin de kleuren van de elementen wordt gegeven. Als het kristal in de *STICK* tekenstijl wordt getekend zullen de bollen steeds een witte kleur krijgen toegekend.

4.3.4 Teken van bindingen

De gebruiker heeft de mogelijkheid om bindingen tussen atomen te laten tekenen, in de *STICK* tekenstijl zullen deze bindingen steeds worden getekend. Het tekenen van bindingen wordt gedaan in de *drawBonds* methode van de *Crysdata* klasse. Deze methode zal in eerste instantie een báziercirkel tekenen en het de naam *bez* toekennen, deze cirkel zal gebruikt worden als vorm waarmee een *bevel* zal worden gedaan, later hier meer over. Vervolgens gaan we de lijst van atomen doorlopen per element deze lijst nogmaals doorlopen, op deze manier kunnen we de afstand tussen elk element berekenen. Voor we deze afstand gaan berekenen gaan we eerst twee testen doen om te voorkomen dat het programma dubbel werk doet. Eerst gaan we na of de twee atomen dezelfde zijn, in dit geval zou het nutteloos zijn een binding te tekenen. De tweede test gaat in de lijst van getekende objecten kijken of er al een binding bestaat tussen deze twee atomen, zo vermijden we dat we de binding tweemaal tekenen. De *bpy.data* module van de Blender API geeft ons toegang tot alle interne data van de Blender omgeving. Door *bpy.data/objects* op te roepen krijgen we een dictionary waarin alle objecten kunnen worden opgeroepen aan de hand van hun ID. Doordat we onze bindingen een naam geven gebaseerd op de atomen die gebonden worden hebben we zo een manier om reeds bestaande bindingen te vinden.

De gebruiker kan zelf de maximale afstand instellen die tussen twee atomen mag zijn om er een binding tussen te tekenen. De afstand tussen twee atomen gaan we berekenen met de formule:

$$d = \sqrt{(x_{a1} - x_{a2})^2 + (y_{a1} - y_{a2})^2 + (z_{a1} - z_{a2})^2}$$

Met *a1* de orthogonale coördinaten van het eerste atoom en *a2* die van het tweede.

Enkel wanneer deze afstand kleiner is dan diegene die de gebruiker kiest, moeten we een binding tekenen tussen de atomen. Hiervoor wordt de *makeBond* methode opgeroepen met de twee te binden atomen als argument. Deze methode gaat eerst met de *bpy.data* module de twee bollen zoeken die de atomen voorstellen. Deze twee objecten zullen dienen als argumenten van de volgende methode die we gaan oproepen, *hookCurve*. In deze methode worden enkele, eerder abstracte, methodes van de *bpy* module gebruikt, daarom zal deze stap voor stap worden doorlopen.

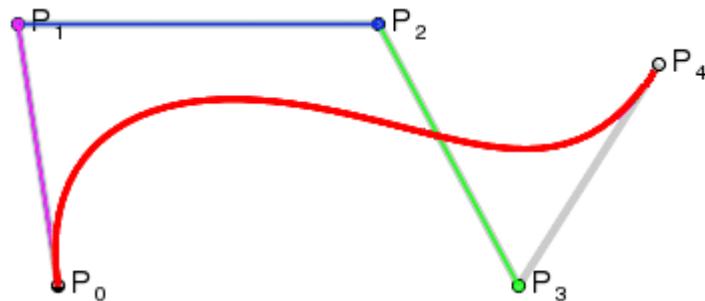
```
curve = bpy.data.curves.new("link", 'CURVE') 531
curve.dimensions = '3D' 532
```

Eerst wordt er een *curve*, of kromme, aangemaakt. Deze krijgt de naam *link* en het type *CURVE* toegewezen. We zeggen ook meteen dat de kromme driedimensionaal is, dit zorgt anders later voor problemen. Merk op dat we onze nieuwe kromme niet in de lijst van objecten aanmaken, we creëren in feite een nieuwe vorm die we later als object kunnen aanmaken.

```
spline = curve.splines.new('BEZIER')
```

533

Vervolgens gaan we nieuwe splines toevoegen aan de kromme. Splines zijn in feite lijnsegmenten waarmee een kromme kan worden voorgesteld, in ons geval gaan we werken met slechts één spline die van het type *bézier* is. Dit zal van onze kromme een *béziers*kromme maken. Een *béziers*kromme is een manier waarop lijnen wiskundig kunnen worden voorgesteld aan de hand van twee of meer punten, die de graad van de *béziers*kromme bepalen volgens: $graad = n - 1$. Figuur[4.1] geeft een voorbeeld van een kromme die bepaald wordt door vier punten. Omdat we in ons geval enkel met rechte lijnstukken gaan werken volstaat een *béziers*kromme van de eerste graad en hoeven we niet dieper in te gaan op de wiskundige berekening van de vorm.



Figuur 4.1: Voorbeeld van een *béziers*kromme met graad 3 (Tregoning, 2007)

Momenteel bestaat onze spline uit één *bézier*punt, we moeten er dus nog één toevoegen om een lijn te bekomen.

```
spline.bezier_points.add(1)                                535
p0 = spline.bezier_points[0]                            536
p1 = spline.bezier_points[1]                            537
```

We kennen onze *bézier*punten toe aan variabelen zodat deze later makkelijk kunnen worden opgeroepen.

```
obj = bpy.data.objects.new("link", curve)                544
m0 = obj.modifiers.new("alpha", 'HOOK')                  545
m0.object = o1                                         546
m1 = obj.modifiers.new("beta", 'HOOK')                  547
m1.object = o2                                         548
```

Nu gaan we van onze kromme een object maken in de Blender omgeving. Aan zo een object kunnen we vervolgens *modifiers* toevoegen. In ons geval gaan we gebruik maken van de *hook modifier*, deze zal het object vasthechten aan een ander object. We maken een nieuwe *hook modifier* aan en noemen deze *alpha*. We hechten deze aan *o1*, het eerste atoom, vast. Dit doen we vervolgens nogmaals voor *o2*, het tweede atoom, en geven deze de naam *beta*. Onze 'haken' hangen nu vast aan onze atomen, maar nog niet aan onze kromme.

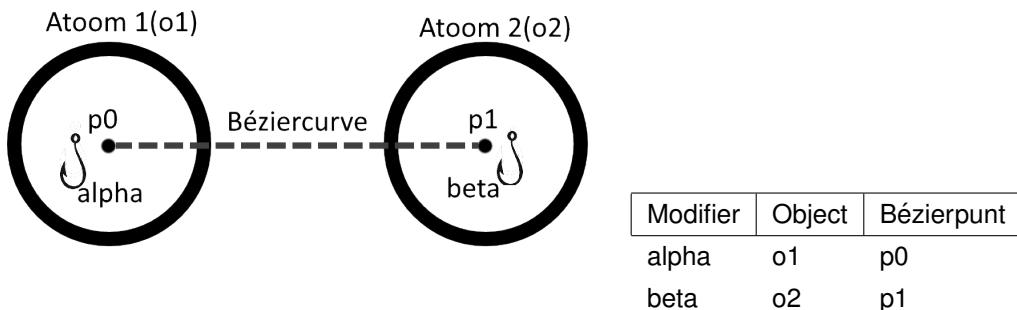
```
bpy.context.collection.objects.link(obj)                550
bpy.context.view_layer.objects.active = obj            551
```

	552
bpy.ops.object.mode_set(mode='EDIT')	553

We gaan eerst onze kromme toevoegen aan onze collectie, hierna kunnen we onze kromme als actief object selecteren. Dit is nodig omdat we in de volgende lijn de tekenmode van Blender gaan veranderen naar de *EDIT* mode. In deze mode worden alle hoekpunten van het actieve object zichtbaar en kan de geometrie van deze in meer detail worden aangepast. We zullen de *EDIT* mode gebruiken zodat we in de volgende stap de twee bázierpunten van onze kromme apart kunnen selecteren.

	560
p0.select_control_point = True	561
p1.select_control_point = False	562
bpy.ops.object.hook_assign(modifier="alpha")	

Herinner dat we eerder onze twee bázierpunten aan de variabelen *p0* en *p1* hebben toegekend. Deze variabelen kunnen we nu gebruiken om de punten op een eenvoudige wijze te (de)selecteren. Eerst gaan we *p0* selecteren en *p1* deselecteren. Dan kunnen we *p0* vasthangen aan de eerste *hook modifier* die we *alpha* hebben genoemd. Dit gaan we herhalen voor *p2* en *beta*. Figuur[4.2] toont een overzicht van welke *modifier* welke objecten vasthecht.



Figuur 4.2: Overzicht van de *hook modifier*

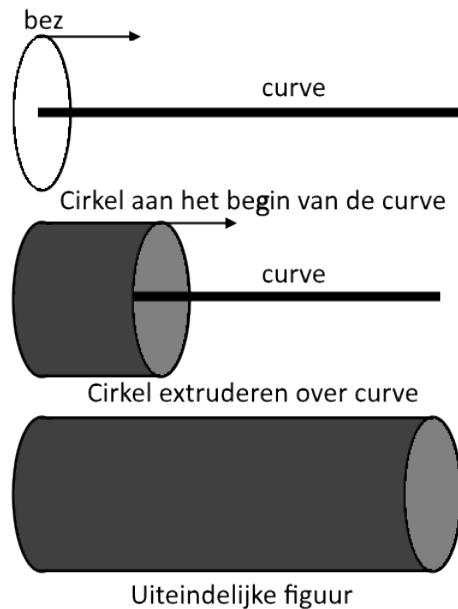
Het vasthechten van objecten zal steeds gebeuren in het originepunt van een object. Als dit punt niet wordt verplaatst zal het zich steeds in het midden van een object bevinden. Hierdoor zal de kromme steeds gebonden zijn aan het midden van de bollen.

Hoewel we nu een kromme hebben getekend tussen onze atomen, gaan we deze nog niet kunnen zien. Omdat een kromme geen volume heeft, zal deze niet zichtbaar zijn op het scherm. Er is nog één laatste stap die we moeten volgen om onze bindingen te tekenen, een *bevel*.

	519
bpy.context.object.data.bevel_object = bpy.data.objects["bez"]	519
]	

In het begin van deze sectie hebben we een báziercirkel, *bez* getekend. Deze gaat nu van pas komen, we gaan namelijk een *bevel* uitvoeren. Een *bevel*, ook wel een *sweep* genoemd, een term uit de wereld van het computertekenen die ruwweg betekent: een driedimensionale vorm creëren door een bepaalde vorm over een pad te extruderen. Op Figuur[3.4] wordt een *bevel* gedaan van

een cirkel over een recht lijnstuk waardoor een cilinder wordt gecreëerd. In realiteit zal er in Blender geen nieuw object worden aangemaakt bij een *bevel*, het is een eigenschap van de kromme die het zijn driedimensionale structuur geeft.



Figuur 4.3: Eenvoudig voorbeeld van een *bevel*

Nu we de binding succesvol kunnen tekenen hoeven we ze enkel nog een naam en kleur te geven. Voor de kleur van de bindingen gebruiken we wit, zo zijn ze duidelijk zichtbaar in het kristal.

4.4 Ontwerpen van een Blender add-on

4.4.1 Add-on header

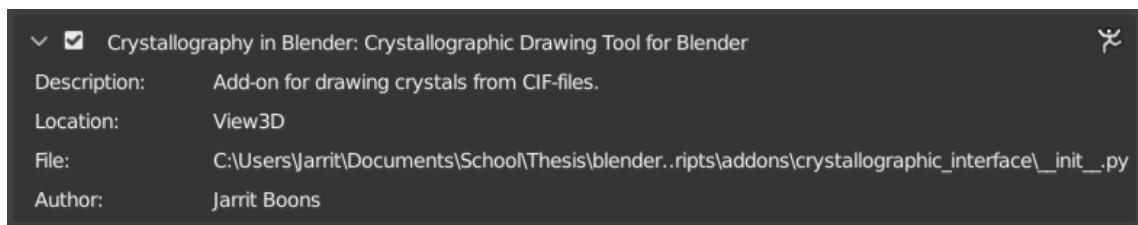
In onze code schrijven we een header, als een dictionary met de naam *bl_info*, waarin we de nodige informatie over de add-on zetten, zie Listing[4.12]. Figuur[4.4] toont aan hoe onze add-on wordt afgebeeld in de *user_preferences* onder het *add-on* tabblad. We merken op dat beide het *name* veld en het *category* veld van de header gebruikt wordt om de add-on initieel weer te geven. De andere velden worden pas zichtbaar als er op het pijltje linksboven wordt gedrukt. We moeten dus de naam en categorie van onze add-on dusdanig kiezen dat een gebruiker meteen weet waarvoor ze dient.

```
bl_info = {
    "name": "Crystallographic Drawing Tool for Blender",
    "description": "Add-on for drawing crystals from CIF-files.",
    "author": "Jarrit Boons", 103
} 104
105
106
```

"blender": (2, 80,0),	107
"location": "View3D",	108
"category": "Crystallography in Blender"	109
}	110

Listing 4.12: De bl_info header van onze add-on

Als naam is er voor *Crystallographic Drawing Tool for Blender*, afgekort naar *CDTB*, gekozen. De categorie spreekt voor zichzelf, en in de beschrijving van de add-on wordt er vermeld dat het programma met CIF-bestanden werkt. De locatie van de add-on is het *View3D* venster, omdat hier het kristal zal getekend worden.

**Figuur 4.4:** Onze add-on afgebeeld in de *user.preferences*

4.4.2 Schrijven van operatoren

Zoals in hoofdstuk drie wordt beschreven gaan we twee operatorklassen maken. Met de eerste operator willen we een bestandsbrowser openen waarin we het CIF-bestand kunnen selecteren dat we willen tekenen.

def invoke(self, context, event):	129
	130
context.window_manager.fileselect_add(self)	131
return {'RUNNING_MODAL'}	132

Listing 4.13: De invoke methode van de eerste operator

Deze klasse geven we een methode *invoke*, zie Listing[4.13], deze zal ervoor zorgen dat zodra de operator wordt uitgevoerd er een filebrowser opent, met de *return* op lijn 132 zorgen we ervoor dat deze operator blijft lopen zolang de bestandsbrowser is geopend. Wanneer we een bestand selecteren, zal de *execute* methode het pad naar het bestand aan een globale variabele toekennen, zodat deze leesbaar is vanuit heel het programma, en stopt de operator.

Bij het maken van de tweede operatorklasse komt iets meer werk kijken. Eerst moeten we alle attributen aanmaken die we later in het paneel willen kunnen aanpassen. Dit doen we in de *register* methode, met behulp van de *bpy.props* module. Met deze module gaan we de attributen aanmaken die moeten geïnterpreteerd worden door het paneel. Een voorbeeld van de creatie van zo een attribuut zien we in Listing[4.14].

bpy.types.Scene.bond_distance = bpy.props.FloatProperty(210
--	-----

```

        name="Bond distance" , 211
        description="Set max distance for bonds to occur" , 212
        default=2, 213
        min=0.0 , 214
        max=10.0 , 215
        precision=2 216
    )
 217

```

Listing 4.14: Het initialiseren van het *bond_distance* attribuut

Met het attribuut in Listing[4.14] zal de gebruiker de maximale bindingsafstand kunnen kiezen, omdat deze afstand een kommagetal mag zijn gebruiken we een *FloatProperty*. Deze functie krijgt enkele argumenten meegegeven:

- name: de naam die in het paneel zal worden weergegeven
- description: een beschrijving van het attribuut welke te zien is als de cursor over het attribuut hangt
- default: de waarde die het attribuut heeft wanneer de add-on wordt ingeladen
- min/max: de uiterste waarde die de gebruiker kan invullen
- precision: het aantal getallen na de komma dat het attribuut kan hebben

Wanneer het attribuut een *IntProperty* of een *BoolProperty* is zullen sommige van deze parameters, zoals de precisie, niet van toepassing zijn. Enumeraties werden in hoofdstuk drie reeds besproken. Attributen van het type *EnumProperty* kunnen enkel de waarde hebben van een van de elementen van de enumeratie die wordt meegegeven als parameter *item*.

De *execute* methode van de tweede operator gaat aanvankelijk, net zoals de eerste operator, zijn attributen toekennen aan globale variabelen zodat ze leesbaar zijn vanuit heel het programma. Hierna roept de operator de functie (*drawCrystal*) op, en zal de code die we bespraken in sectie twee en drie worden uitgevoerd.

4.4.3 Creëren van de user interface

Als link tussen onze add-on en de gebruiker gaan we gebruik maken van de *Panel* klasse van Blender. Bij aanmaken van een paneel moeten we deze eerst enkele belangrijke attributen toekennen, zie Listing[4.15].

```

bl_idname = "CDTB_Panel" 274
bl_label = "CDTB_Panel" 275
bl_space_type = "VIEW_3D" 276
bl_region_type = "TOOLS" 277
bl_context = "objectmode" 278

```

```
bl_category = "CDTB"
```

279

Listing 4.15: Enkele belangrijke attributen van de paneelklasse

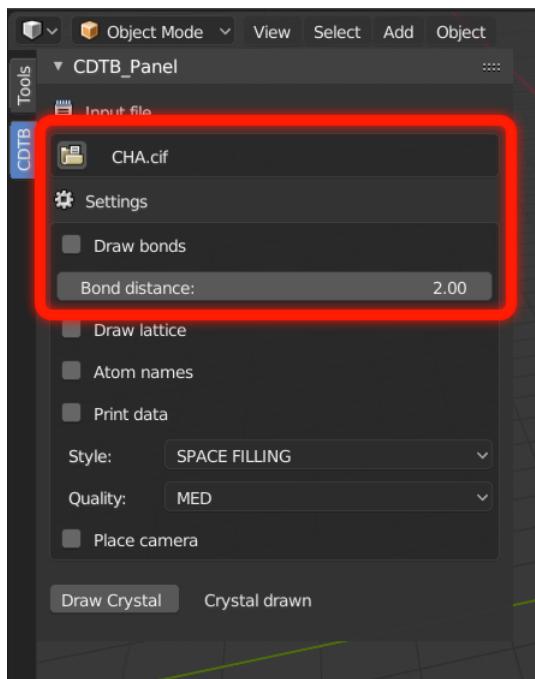
Met deze attributen bepalen we hoe en waar we de add-on willen weergeven, en welke naam er bovenaan het paneel staat. In de *draw* methode gaan we de layout van het paneel personaliseren. Om de add-on gebruiksvriendelijk te houden gaan we de opmaak van het paneel beperken tot enkele vakken, symbolen en gaan we geen specifieke kleuren toekennen.

De paneelklasse heeft een attribuut *layout* waarop we verschillende methodes kunnen oproepen. Telkens we zo een methode oproepen wordt er een onderdeel toegevoegd aan het paneel. Deze methodes kunnen we ook oproepen op de onderdelen van het paneel, zo kunnen we ze verder onderverdelen. Listing[4.16] toont een deel van de *draw* methode waarin we het paneel opbouwen door een reeks van deze methodes in een bepaalde volgorde op te roepen. Het blok code van Listing[4.16] komt overeen met het deel van paneel dat zich in de rode kader bevindt op Figuur[4.5]. Op deze manier kunnen we de code stap voor stap vergelijken met ons uiteindelijke paneel.

box = layout.box()	294
row = box.row()	295
splitrow = row.split(factor=0.075)	296
left_col = splitrow.column()	297
right_col = splitrow.column()	298
left_col.operator('error.scan_file',icon_value=108,text="")	299
right_col.label(text=file_path.rsplit(dir_sep, 2)[-1])	300
layout.label(text = 'Settings',icon_value =117)	301
box = layout.box()	302
box.prop(scn,'draw_bonds')	303
box.prop(scn,'bond_distance')	304

Listing 4.16: Het opbouwen van een paneel

Op de eerste lijn van Listing[4.16] maken we een vak aan met de *box* methode, dit zal een kader tekenen rond alles wat zich in dit vak bevindt. Hierna maken we een rij aan in het vak door de *row* methode op de variabele *box* op te roepen. Deze rij splitsen we op en verdelen deze in twee kolommen, waarvan de linker slechts 7.5% van de volledige breedte van het paneel is. Op de linkerkolom roepen we de methode *operator* op, lijn 299. Hiermee plaatsen we een drukknop die de operator uitvoert waarmee we een bestand kunnen inlezen. We zetten geen tekst op de drukknop maar gebruiken het icoon van een map, deze heeft 108 als ID. In de rechterkolom plaatsen we een label waarop de globale variabele van het pad naar het bestand wordt geschreven. Omdat het volledige pad niet in het paneel past, laten we alle tekens voor het laatste scheidingsteken vallen zodat we enkel de bestandsnaam en extensie overhouden. Dit is het laatste dat we toevoegen in dit vak van het paneel.



Figuur 4.5: Het paneel (onderdelen uit Listing[4.16] in rode kader)

Vooraleer we een nieuw vak gaan aanmaken, voegen we een label toe en geven dit zowel de naam *settings* als een icoon. Dit label is de titel van het volgende vak, wat niet alleen esthetisch aangenaam is, maar ook overzichtelijk is voor de gebruiker.

Hierna maken we een nieuw vak aan waarin we alle tekenopties gaan geven. Deze opties zijn de attributen van de tweede operatorklasse en geven we weer door de *prop* methode op de roepen op het vak, en de naam van het attribuut als parameter mee te geven. Een *BoolProp* zoals *draw_bonds* zal worden weergegeven als een selectievakje, en *bond_distance*, dewelke een *FloatProp* is, zal in ons paneel als een invoerveld en schuifbalk worden weergegeven.

4.5 Conclusie

In dit hoofdstuk hebben we eerst bekeken hoe we de OpenBabelGUI moeten installeren op een Windowssysteem, zodat we dit in ons programma kunnen oproepen als subprocess. Hierna hebben we ons verdiept in de installatie van de CifFile module van PyCIFRW. Deze kunnen we installeren op een systeem aan de hand van een Python3.7 installatie en de pip installer. Deze stap kunnen we overslaan door de map met de CifFile module meteen in de Pythoninstallatie van Blender te plaatsen.

Na dieper in te gaan op de installatie van deze externe programma's hebben we de onderliggende code van ons programma onder de loep genomen en de werking van enkele belangrijke functies overlopen. We zagen onder andere hoe we controleren of een bestand een CIF-bestand is en hoe we met de subprocess module van Python OpenBabel als extern process kunnen oproepen om

het CIF-bestand te converteren naar één zonder inwendige symmetrie.

We hebben kort aangehaald hoe het is om zelf een CIF-parser te ontwerpen en merkten dat dit, hoewel doenbaar, erg veel werk is. Aangezien dit niet de essentie van het onderzoek is, zijn we hier niet verder op ingegaan.

Vervolgens hebben we bekeken hoe we met behulp van de CifFile module van PyCIFRW CIF-bestanden kunnen parsen. We hebben een voorbeeld bekeken waarin we de roosterparameters van een kristal aan de variabelen van de Cell toekennen, en hoe de lijst van atomen kan verkregen worden door het lusblok van een CIF-bestand te doorlopen. Omdat OpenBabel nieuwe atomen geen eigen ID geeft, hebben we zelf een manier gevonden die met een lijst en een teller elk atoom van het kristal een unieke ID geeft.

Het tekenen in Blender doen we met behulp van de bpy module die de Blender API aanbiedt. Deze module hebben we onder andere gebruikt om de omkadering van het eenheidskristal te tekenen. Dit doen we door eerst de hoekpunten te tekenen als bollen en deze nadien met elkaar te verbinden met behulp van cilinders. We zagen ook hoe we een materiaal aan een object moeten toekennen zodat we het een kleur kunnen geven.

Om de kleur en de straal van een atoom te weten gaan we gebruik maken van twee dictionaries die zich in externe bestanden bevinden. Deze kunnen we eenvoudig inlezen met de eval methode. Aan de hand van de conversiematrix kunnen we de orthogonale coördinaten van elk atoom berekenen, zo kunnen we elk element uit de lijst van atomen tekenen als een bol met een bepaalde positie, straal en kleur.

Om de bindingen tussen atomen te tekenen moeten we eerst weten of deze zich dicht genoeg bij elkaar bevinden, hiervoor gaan we tussen elk element de afstand berekenen. Pas als deze afstand kleiner is dan een bepaalde waarde en als we nog geen binding hebben getekend tussen de atomen zal de binding worden getekend.

Bij het tekenen van een nieuwe binding gaan we eerst een kromme aanmaken. Uit deze kromme creëren we een nieuwe bázierkromme, dit is een lijn bestaande uit segmenten die op een wiskundige manier kan worden beschreven. Deze kromme krijgt op elk uiteinde één bázierpunt toegekend. We hangen aan onze kromme alvast twee hook modifiers, één voor elk atoom dat we willen binden. Met de edit mode van Blender kunnen we de Bázierpunten van de kromme apart aanspreken, zo kunnen we het ene punt aan het ene atoom, en het andere punt aan het andere atoom hangen. Door een bevel over de kromme te doen verkrijgen we een cilindervormige binding die vasthangt aan beide atomen.

In laatste sectie van dit hoofdstuk hebben we een Blender add-on gemaakt. Zo een add-on bestaat steeds uit een header, hierin schrijven we informatie over onze add-on. Onze add-on heeft twee operator klassen, de eerste operator zal een bestandsbrowser openen waarin de gebruiker een bestand kan selecteren. De tweede operator heeft verschillende attributen die moeten aangemaakt worden in de register methode van de operatorklasse. We geven de attributen hier een naam, een beschrijving en eventueel nog andere eigenschappen. De execute methode van de tweede operator zal al deze attributen toekennen aan globale variabelen en de hoofdfunctie van heel het programma oproepen. Deze functie zal de kristaldata converteren, inlezen en tekenen.

Om de add-on zichtbaar te maken voor een gebruiker maken we een paneelklasse aan. Deze klasse bevat enkele attributen waarmee we onder andere de naam van het paneel kunnen bepalen. In de draw methode kunnen we het paneel opbouwen door methodes van het layout attribuut op te roepen. Met deze methodes kunnen we onder andere vakken, rijen en kolommen aanmaken. Op het paneel kunnen we ook de attributen van de tweede operator weergeven, zodat deze kunnen aangepast worden door de gebruiker. Ten slotte kunnen we aan het paneel drukknoppen toevoegen die de operatoren, en dus ons programma, zal uitvoeren.

Hoofdstuk 5

Testen en Resultaten

In dit hoofdstuk worden de resultaten van het programma besproken. Eerst zal er, aan de hand van een stappenplan, worden bekeken hoe de add-on moet worden geïnstalleerd en hoe deze te gebruiken.

Er zal ook een vergelijking worden gedaan tussen een bestaand kristalvisualisatieprogramma en het programma als kristalvisualisatietool in Blender. Dit zal onder andere gedaan worden op basis van snelheid, gebruiksvriendelijkheid en uitbreidbaarheid.

Er worden in dit hoofdstuk ook enkele valkuilen besproken die in de loop van dit onderzoek zijn opgedoken, en hoe deze worden opgelost. Sommige van deze valkuilen vormen echter grote problemen die in Blender moeilijk of niet kunnen worden vermeden. Er zal vermeld worden hoe deze problemen in dit onderzoek al dan niet zullen worden omzeild. Ten slotte zal er een blik worden geworpen op hoe het programma in de toekomst kan worden verbeterd en uitgebreid.

Net zoals in hoofdstuk vier zal er voornamelijk gebruik worden gemaakt van de actieve schrijfwijze en de wetenschappelijke 'we'-vorm.

In de eerste sectie staat een stappenplan dat het hele proces beschrijft dat moet gevuld worden om met de add-on een kristal te tekenen in Blender. In de tweede sectie zullen resultaten van het programma worden beschreven. Hier worden onder andere enkele getekende kristallen weergegeven. In de derde sectie worden deze resultaten vergeleken met die van een ander kristalvisualisatieprogramma.

De vierde sectie zal de valkuilen bespreken waarmee dit onderzoek had te maken. In deze sectie worden ook enkele tekortkomingen van het programma gezien. In de vijfde, en voorlaatste, sectie van dit hoofdstuk wordt er gekeken naar wat de toekomst te bieden heeft voor het programma. Ten slotte zal er een conclusie gegeven worden waarin de belangrijkste punten van dit hoofdstuk nogmaals aan bod komen.

5.1 Een kristal tekenen met de add-on

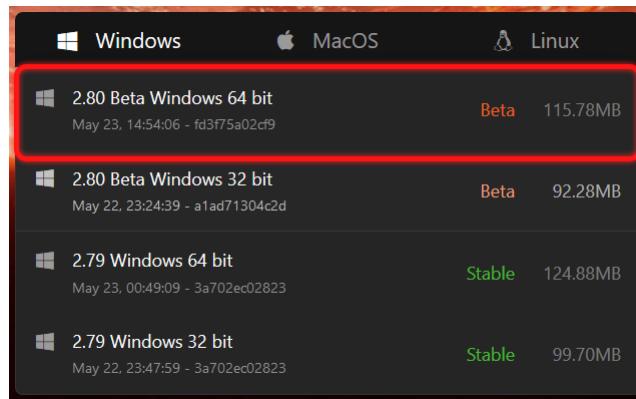
Deze sectie dient als een handleiding die een gebruiker kan volgen om onze add-on te installeren in Blender en ze te gebruiken. Dit gaan we doen in de vorm van een stappenplan.

Dit is het stappenplan voor de installatie en het gebruik van de add-on op het besturingssysteem Windows. De add-on moet werken op andere besturingssystemen maar de installatie van de add-on op deze gaan we niet bekijken.

5.1.1 Stap 1: Blender downloaden

Als eerste moeten we Blender downloaden op ons systeem. Onze add-on is gemaakt voor versie 2.8x van Blender. In deze versie is er veel vernieuwd aan de Blender API waardoor de add-on niet achterwaarts compatibel is. Dit wil zeggen dat de add-on niet zal werken op oudere versies van Blender zonder de code aan te passen. In dit hoofdstuk spreken we, tenzij anders vermeld, altijd over de 2.8x versie van Blender.

Blender kan gedownload worden op de downloadpagina van hun officiële website: <https://builder.blender.org/download/>. Hier kiezen we het besturingssysteem en de bit-versie van ons systeem, en selecteren we de nieuwste versie van Blender, rode kader op Figuur[5.1]. Dit download het ZIP-bestand waarin de Blender installatie staat. Als de download voltooid is, kunnen we het ZIP-bestand uitpakken op ons systeem. Dit creëert een map waarin de Blender installatie staat. In deze map vinden we het uitvoerbare bestand *blender.exe*, waarmee we Blender opstarten. Dit is tevens ook de map waarin we de CifFile module in plaatsen, zie eerste sectie van hoofdstuk vier.



Figuur 5.1: Overzicht van Blenderversies op downloadpagina van blender

5.1.2 Stap 2: Externe programma's installeren

Om de add-on te laten werken moeten we de OpenBabelGUI en de CifFile module van PyCIFRW op ons systeem installeren. We kunnen deze programma's downloaden op volgende webpagina's:

- OpenBabelGUI: <http://openbabel.org/wiki/Category:Installation>

- PyCIFRW: <https://pypi.org/project/PyCifRW/#description>
- CifFile module: <https://github.com/JarritB/Thesis/tree/master/>

Hoe we deze programma's werkende krijgen op ons systeem wordt uitgelegd in de eerste sectie van hoofdstuk vier.

5.1.3 Stap 3: Add-on downloaden

Een ZIP-bestand van de add-on kan worden gekopieerd uit de bijlagen van deze tekst.[Bijlage G]

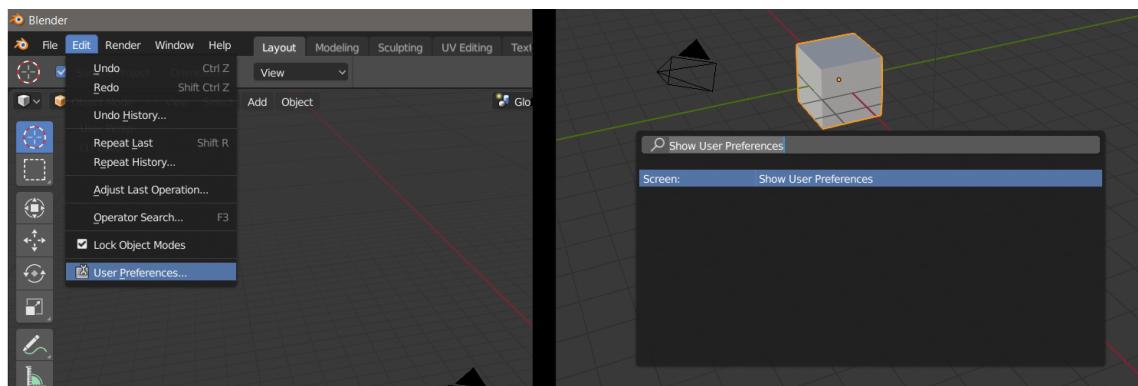
De meest recente versie van de add-on vinden we op de GitHub pagina van dit onderzoek: <https://github.com/JarritB/Thesis/tree/master/blender-2.80.0-git.3c8c1841d72-windows64/2.80/scripts/addons> in de map *crystallographic_interface*. Deze map bevat de code en dictionaries van onze add-on.

We downloaden de map en plaatsen deze in de *addons* submap van de Blender installatie. (*blender-2.80.0-git.3c8c1841d72-windows64 >> 2.80 >> scripts >> addons*)

5.1.4 Stap 4: Add-on activeren

Om de add-on te activeren moeten we eerst Blender opstarten door *blender.exe* uit te voeren. Vervolgens gaan we het venster met de *user preferences* openen. Dit kunnen we doen op twee manieren:

- In de linkerbovenhoek bij de optie *Edit* (links op Figuur[5.2])
- Zoekfunctie openen (F3-toets), en *Show User Preferences* intypen (rechts op Figuur[5.2])



Figuur 5.2: Openen van de *User Preferences* via GUI(links) en Zoekfunctie(rechts)

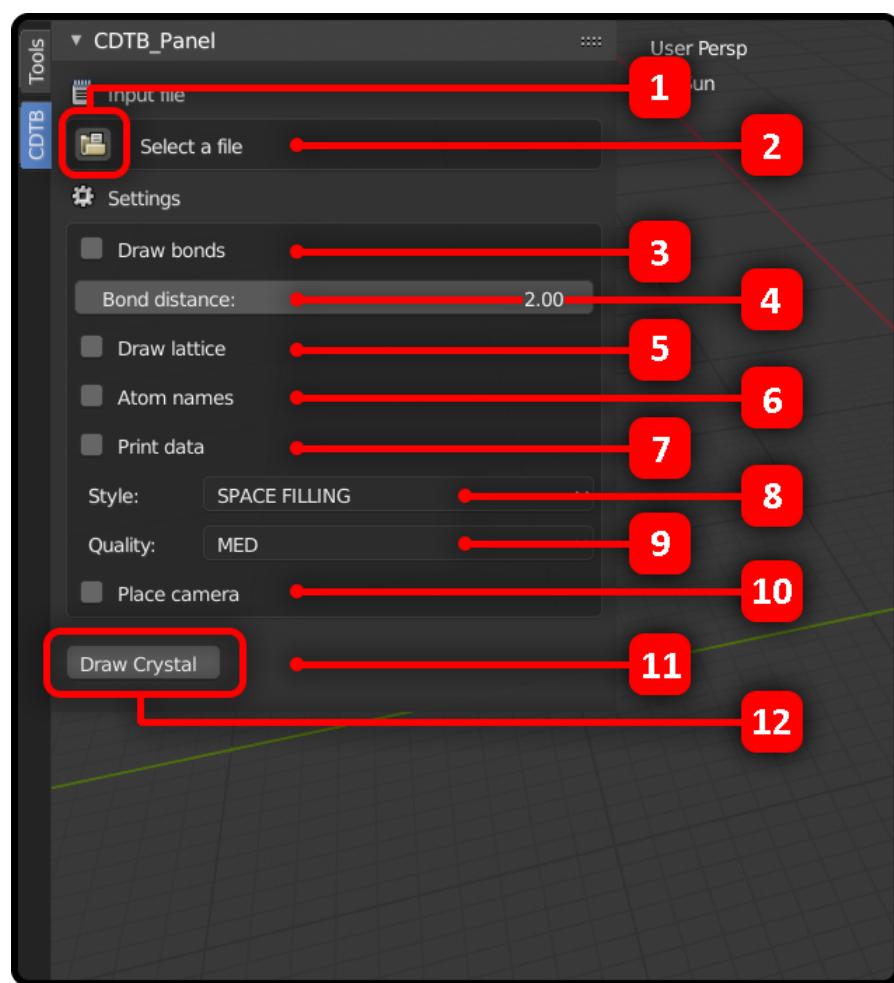
Onder de tab *add-ons* vinden we een lijst met add-ons die we kunnen gebruiken in Blender. Omdat we onze add-on in de map van Blender hebben geplaatst zou deze in de lijst moeten te vinden zijn onder de naam: *Crystallography in Blender: Crystallographic Drawing Tool for Blender*. Om de

add-on te activeren hoeven we enkel het selectievakje links van de naam aan te duiden. Als het vakje is aangevinkt is de add-on geactiveerd en klaar voor gebruik.

5.1.5 Stap 5: Kristal tekenen met de add-on

We kunnen het label van onze add-on zien onder het veld *Tools* op het *3D View* venster. Dit is het grote venster in het midden dat we zien wanneer we Blender opstarten. Het *Tools* veld vinden we aan de linkerkant van dit venster, maar kan verborgen zijn. Met de T toets tonen en verbergen we tools veld. Bij het aanklikken van het *CDTB* label zal ons paneel tevoorschijn komen. Als we ons in 'Edit Mode' bevinden zal het label niet getoond worden. Door de Tab toets in te drukken geraken we terug in 'Object Mode', waar onze add-on zou moeten staan.

Om een kristal te tekenen moeten we eerst een CIF-bestand selecteren. Dit doen we door met het icoon naast "Select a file" een bestandsbrowser te openen en ons CIF-bestand te selecteren. Dan hebben we de mogelijkheid om een aantal tekenvoorkeuren aan te passen, in Tabel[5.1] worden deze uitgelegd. Als we de voorkeuren hebben aangepast hoeven we enkel nog op de *Draw Crystal* knop te klikken, en te wachten tot het kristal getekend is. Dit duurt, afhankelijk van de gekozen tekenopties en de grootte van het kristal, enkele seconden tot enkele minuten.



1	Bij indrukken: opent een bestandsbrowser waar het CIF-bestand kan geselecteerd worden
2	Toont het geselecteerde CIF-bestand
3	Als aangeduid: tekent bindingen tussen de atomen
4	Getal: bepaalt maximale afstand waartussen twee atomen worden gebonden
5	Als aangeduid: tekent omkadering van de eenheidscel
6	Als aangeduid: toont namen van de atomen op de tekening
7	Als aangeduid: drukt de kristalinformatie af in de Blender terminal
8	Keuzelijst: tekent kristal in gekozen stijl STICK: toont enkel bindingen; BALL AND STICK: toont verkleinde atomen, bindingen zichtbaar; SPACE FILLING: toont atomen op ware grootte, bindingen meestal niet zichtbaar
9	Keuzelijst: bepaalt kwaliteit van de vormen, hogere kwaliteit verlengt tekenduur
10	Als aangeduid: plaats een camera en belichting, voor het renderen(F12) van het kristal
11	Bij indrukken: visualiseert het CIF-bestand
12	Toont opmerkingen aan de gebruiker

Tabel 5.1 Overzicht van de instellingen van de add-on

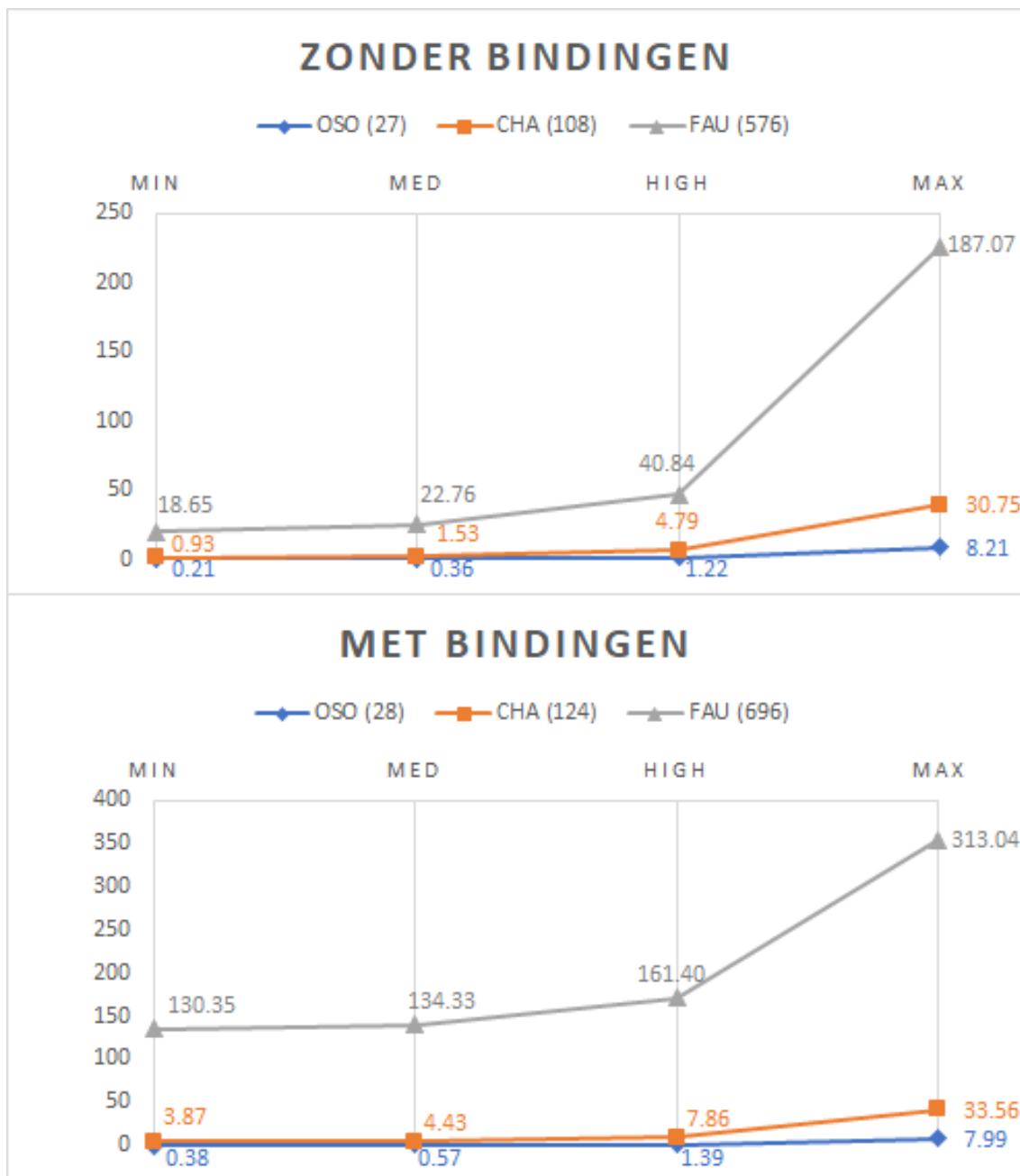
5.2 Resultaten

5.2.1 Snelheid

In een gebruiksvriendelijk programma speelt de snelheid ervan een belangrijke rol. We willen de tijd die een gebruiker moet wachten beperken, maar niet ten koste van de kwaliteit van de uitvoer. In dit deel van de sectie gaan we kijken naar de duur van het programma bij verschillende parameters en welk deel van het programma het grootste aandeel van de tijd opeist. Merk op dat snelheid vaak evenredig is met de specificaties van het systeem. De specificaties van het systeem waarop we de testen gaan uitvoeren, zijn te vinden in de bijlagen.[Bijlage H]

Eerst gaan we de totale duur van het programma testen bij het tekenen van kristallen. Deze testen worden gedaan voor drie kristallen met verschillende grootte. De resultaten van deze testen worden weergegeven op de grafieken in Figuur[5.3]. Enkele opmerkingen bij deze grafieken zijn:

- Op de verticale as wordt de looptijd in seconden weergegeven
- Op de horizontale as worden de verschillende tekenkwaliteiten weergegeven waartussen de gebruiker kan kiezen
- De tekenstijl *LOW* hebben we niet getest
- Het verhogen van de tekenkwaliteit gebeurt met factor vier per stap, en begint bij 32 segmenten
- De getallen in de legende van de eerste grafiek zijn het aantal getekende atomen
- De getallen in de legende van de tweede grafiek zijn het aantal getekende bindingen, het aantal atomen is gelijk aan de waarden in de eerste grafiek
- De gegeven waarden zijn telkens het gemiddelde van drie testen
- Deze resultaten zijn voor het tekenen van het "BALL AND STICK"-model.
- Hoewel de andere kristalvoorstellingen ook werden getest, kregen we gelijkaardige waarden aan die van het "BALL AND STICK"-model en worden deze niet gepubliceerd
- Bij het testen wordt enkel de kristalomkadering getekend, andere instellingen staan uit



Figuur 5.3: Duur van het programma in functie van gekozen kwaliteit

Aan de hand van deze resultaten kunnen we er vanuit gaan dat de duurtijd evenredig is met het aantal atomen dat moet getekend worden. We merken op dat dit geen recht evenredig verband is, het kristal FAU bestaat uit ongeveer vijf keer meer atomen dan CHA, terwijl het tekenen van FAU een factor 20 langer duurt. Dit kan het gevolg zijn van het stelselmatig vol geraken van het werkgeheugen, wat kan leiden tot vertraging van het systeem.

Het aantal bindingen dat wordt getekend heeft ook een grote impact op de snelheid. We stellen

vast dat het tekenen van de bindingen een vaste tijd duurt, ongeacht de tekenkwaliteit. Dit is omdat de tekenkwaliteit enkel invloed heeft op het aantal segmenten van de atomen. We kunnen dit grafisch waarnemen doordat de grafieken op Figuur[5.3] ruwweg dezelfde vorm tonen maar opwaarts verschoven zijn. Door deze vaste duur zal het aandeel aan tijd dat het tekenen van de bindingen in beslag neemt, groter zijn bij een lagere kwaliteit. Bij FAU zal het tekenen van de bindingen op de laagste tekenkwaliteit tot wel 85 procent van de totale loopduur van het programma opeisen, terwijl dit bij de hoogste kwaliteit nog maar 40 procent is.

5.2.2 Output van het programma

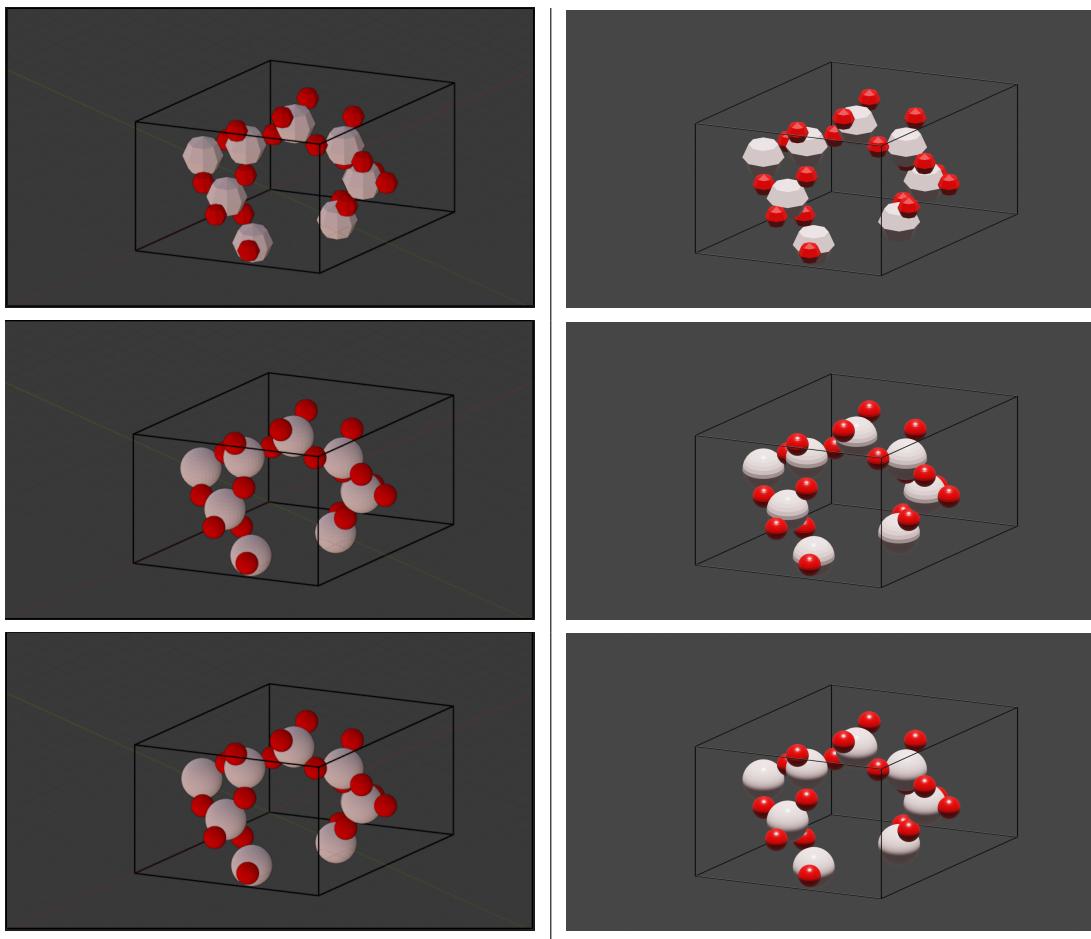
In dit deel kijken we voornamelijk naar hoe ons programma kristallen afbeeldt in Blender. We gaan, met behulp van figuren, bekijken hoe de verschillende modellen eruitzien, hoe we door het verhogen van de kwaliteit een mooier beeld krijgen en wat er gebeurt als we ons kristal gaan renderen.

Op Figuur[5.4] kunnen we de invloed zien van de kwaliteit, i.e. het aantal segmenten, van de atomen. In de linkerkolom staan de tekeningen zoals ze in Blender worden weergegeven en in de rechterkolom vinden we de gerenderde tekeningen. We hebben bewust gekozen de stapgrootte tussen deze voorbeelden constant te houden. Tabel[5.2] toont het aantal segmenten per atoom voor de verschillende tekenkwaliteiten. Door de exponentiële toename zou het verschil tussen de kwaliteiten duidelijk zichtbaar moeten zijn. Dit is ook zo, tot op een bepaald punt. Er is een duidelijk verschil tussen de eerste en tweede afbeelding van de linkerkolom van Figuur[5.4], de twee onderste afbeeldingen zijn echter al moeilijker van elkaar te onderscheiden. En wetende dat er tussen deze twee nog een stap zit, kunnen we vaststellen dat het, in de meeste gevallen, niet noodzakelijk is in de hoogste kwaliteit te tekenen.

Tabel 5.2 Segmenten per atoom bij bepaalde kwaliteit

MIN	32
LOW	128
MED	512
HIGH	2048
MAX	8192

Een voorbeeld van zo een geval waarbij we de hoogste kwaliteit willen, is wanneer we het kristal willen renderen naar een afbeelding. Het concept renderen hebben we reeds besproken in de vijfde sectie van het tweede hoofdstuk. In de rechterkolom van Figuur[5.4] staan de gerenderde versies van de linkerkolom. Nu er belichting aan te pas komt, wordt het verschil in kwaliteit tussen de onderste twee duidelijk. Dit is dan ook de voornaamste reden om in de hoogste kwaliteit te tekenen.



Figuur 5.4: Vergelijking van de tekenkwaliteit voor het kristal OSO (van boven naar onder: MIN MED MAX)

5.3 Vergelijking met VESTA

Met deze resultaten kunnen we een vergelijking maken met een bestaand kristalvisualisatieprogramma. Als programma hebben we gekozen voor VESTA, omdat dit het meest gebruikte en meest performante is.

5.3.1 Snelheid

In Tabel[5.3] wordt de tekensnelheid van ons programma vergeleken met die van VESTA. Dit doen we voor de drie kristallen van Figuur[5.3]. In VESTA worden eerst de berekeningen gedaan van de posities van de atomen en bindingen, dan worden deze pas weergegeven. VESTA geeft echter enkel de tijd die nodig was om de berekeningen te maken. Doordat de kristallen vrijwel ogenblikkelijk verschijnen, is het erg moeilijk de werkelijke tekenduur te bepalen. Het is ook belangrijk op te merken dat VESTA meer atomen tekent per kristal, omdat het ook atomen buiten de eenheidscel tekent, terwijl ons programma enkel de atomen erbinnen tekent. Om de vergelijking eerlijk te

maken zullen we in beide programma's de bindingen tussen de atomen tekenen. De duur wordt weergegeven in seconden en de kwaliteit staat in beide programma's op 512 segmenten per atoom.

Tabel 5.3 Vergelijking in tekenduur tussen ons programma en VESTA, tijd in seconden

	Blender	VESTA	Factor
OSO	.570	0.034	16.67
CHA	4.430	0.062	71.45
FAU	134.330	0.331	405.83

In de laatste kolom wordt vermeld hoeveel maal sneller het tekenen in VESTA wordt gedaan ten op zichte van het tekenen in Blender. Het valt op dat VESTA kristallen veel sneller tekent, bij grote kristallen tot zelfs 400 maal sneller. Als we geen bindingen zouden tekenen met de add-on, zou deze factor nog 70 bedragen, wat een redelijke afname is, maar nog steeds een groot getal is.

Uit deze resultaten kunnen we concluderen dat het tekenen van kristallen met onze add-on veel langer duurt dan in VESTA, en het verschil in duur toeneemt bij het tekenen van grotere kristallen.

5.3.2 Gebruiksvriendelijkheid

Ook op het vlak van gebruiksvriendelijkheid scoort VESTA erg hoog. VESTA is eenvoudig te downloaden, kristallen worden eenvoudig maar duidelijk weergegeven en kan naast het CIF-formaat ook andere formaten visualiseren. Er zijn een groot aantal tekenopties waarmee onder andere de tekenstijl, grenzen en kleuren kunnen worden aangepast. Hiernaast bezit VESTA een aantal handige features waarmee er specifieke informatie over het kristal kan worden berekend en weergegeven. Om deze features te gebruiken heeft de gebruiker wel een bepaalde kennis over het programma nodig. Een laatste en niet onbelangrijke eigenschap van VESTA is dat het volledig gratis is.

We hebben er naar gestreeft onze add-on zo gebruiksvriendelijk mogelijk te maken. Door het eenvoudig maar duidelijke paneel is onze add-on gemakkelijk in omgang, zelfs voor onervaren gebruikers. Het aantal features is eerder beperkt in vergelijking met VESTA. Zo is er, buiten de namen van atomen, geen mogelijkheid tot het weergeven van kristalinformatie en zijn de tekengrenzen beperkt tot binnen de eenheidscel.

Als we kijken naar de weergave van het kristal merken we dat het getekende kristal in Blender veel interactiever is dan in VESTA mogelijk is. In Blender kunnen we individuele atomen verplaatsen, verwijderen en verschalen. Terwijl VESTA deze mogelijkheid niet biedt.

Aan de hand van deze informatie is het moeilijker een winnaar te kiezen op vlak van gebruiksvriendelijkheid dan het was bij de tekensnelheid van de programma's. Dit is deels omdat beide programma's hun voordelen en beperkingen hebben. Veel hangt af van de toepassing van het programma. Voor een meer wetenschappelijke aanpak zullen we VESTA verkiezen, terwijl voor een meer interactieve oplossing onze add-on kan gebruikt worden.

5.3.3 Uitbreidbaarheid

Dit is het vlak waar onze add-on uitblinkt, en de voornaamste reden van dit onderzoek.

Hierboven hebben we VESTA geprezen voor het groot aantal features die het aanbiedt in vergelijking met onze add-on. Nu echter de basis van onze add-on is gelegd, kunnen we deze verder uitbreiden en allerlei features toevoegen. Het zelf creëren van features is met VESTA niet mogelijk. Hoewel VESTA enkele malen per jaar een update krijgt, zijn dit voornamelijk bugfixes en ligt het volledig in de hand van de makers of er nieuwe features worden toegevoegd.

Op vlak van uitbreidbaarheid is onze add-on de grote winnaar. Het toevoegen van features aan de add-on is een kwestie van lijnen code toe te voegen aan de broncode van het programma. In de vijfde sectie van dit hoofdstuk gaan we in meer detail mogelijke uitbreidingen bespreken.

5.4 Valkuilen en problemen

5.4.1 Blender 2.80

Zoals eerder gezegd, werken we in dit onderzoek met Blender versie 2.80. Deze versie van Blender is op het moment van dit onderzoek nog in beta, dit wil zeggen dat Blender, hoewel bruikbaar, constant onder constructie is. Dit kan leiden tot instabiliteit van de software wat een risico vormde voor dit onderzoek. Ondanks het risico, zijn er in de loop van dit onderzoek geen problemen geweest als gevolg van deze instabiliteit.

Doordat er in de 2.80 versie veel is veranderd ten op zichte van eerdere versies, en deze relatief nieuw is, was het geregeld moeilijk correcte informatie te vinden op fora. De Blender API heeft gelukkig een zeer duidelijke documentatie die vaak werd geraadpleegd in dit onderzoek.(Blender, 2019)

5.4.2 PyCIFRW

In de tweede sectie van het vierde hoofdstuk wordt er uitgelegd hoe we gaan controleren of de OpenBabel geïnstalleerd is op het systeem. In het geval dat deze installatie niet wordt gevonden staat er: "Hoewel het kristal niet kan geconverteerd worden, zullen we toch een deel van het kristal kunnen tekenen. Door een fout in de code van de CifFile module verschijnt er een foutbericht als er een te lange padnaam wordt meegegeven bij het inlezen van het CIF-bestand en wordt het programma afgesloten. Doordat de bestandsbrowser steeds het volledige pad meegeeft kan dit probleem zeer moeilijk worden opgelost, en zelfs dan zouden enkel bestanden in de map van Blender kunnen ingelezen worden.

Zodra PyCIFRW een update krijgt, die dit probleem oplost, zou onze add-on moeten werken zelfs zonder installatie van OpenBabel. Tot dan zal het kristal niet kunnen getekend worden zonder OpenBabel.

5.4.3 bpy module

In de vorige hoofdstukken hebben we opgemerkt dat het tekenen van kristallen relatief lang duurt, zeker in vergelijking met VESTA. Dit ligt deels aan het feit dat de bpy module van Blender eerder traag is. Doordat we deze meerdere malen moeten oproepen telkens we een atoom of binding tekenen, duurt het lang voordat het volledige kristal getekend wordt.

Dat tekenen met de bpy module eerder traag is kunnen we bewijzen aan de hand van een test. We schrijven een functie, Listing[5.1], die enkel bollen tekent en meten de tijd dat dit duurt. Door deze tijd te vergelijken met de tijd dat ons programma nodig heeft om een kristal zonder bindingen te tekenen weten we ruwweg hoelang ons programma bezig is met de berekeningen. De resultaten worden in Tabel[5.4] weergegeven. Bij de testen is het aantal segmenten per bol gebruikt dat overeenstemt met de *MED* tekenkwaliteit.

```

import time
import bpy
S = time.time()
for i in range(576):
    bpy.ops.mesh.primitive_uv_sphere_add(segments=32, ring_count=16,
                                          size=0.5, location = (0,0,i))
print(time.time() - S)

```

Listing 5.1: "Testprogramma dat een aantal bollen tekent en de duur meet"

Tabel 5.4 Vergelijking duur van ons programma en het tekenen van enkel bollen, tijd in seconden

# atomen	Programma	Testfunctie	Verschil	%
27	0.36	0.29	0.07	19.4
108	1.53	1.31	0.22	14.3
576	22.76	16.95	5.81	25.5

Uit Tabel[5.4] leiden we af dat ons programma gemiddeld 80 procent van de totale loopduur besteedt aan het tekenen van de atomen. Hoewel dit kan gezien worden als een *bottleneck* van het programma, kunnen we hier weinig of niets aan veranderen en zijn we slechts in staat de overige 20 procent te optimaliseren.

5.5 Uitbreidingen

In deze sectie bespreken we enkele uitbreidingen die in de toekomst nog kunnen worden uitgevoerd. Hoewel de lijst van mogelijke uitbreidingen op zich eindeloos is omdat met de combinatie

van Python modules en de Blender API bijna alles mogelijk is, zullen we ons beperken tot enkele interessante features.

5.5.1 Bepalen van grenzen

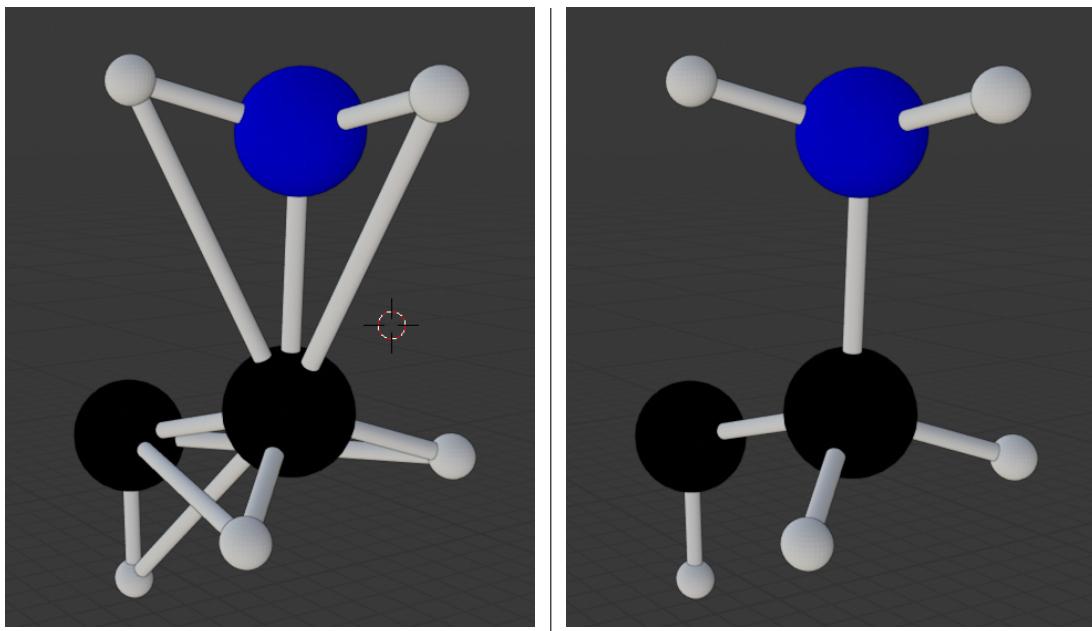
Eén feature dat voorlopig nog niet aanwezig is in de add-on, is de mogelijkheid tot het bepalen van de grenzen tot waar getekend moet worden. Deze grenzen zijn op dit moment de vlakken van het eenheidskristal. Door deze grenzen uit te breiden zal de gebruiker zelf kunnen kiezen hoe groot het getekende kristal mag zijn.

Om deze feature te verwezenlijken zullen we vanuit de inwendige symmetrie van het kristal een lijst moeten maken van alle atomen die binnen deze grenzen liggen. Aangezien OpenBabel deze mogelijkheid niet aanbiedt, zal er moeten gezocht worden naar een programma dat dit wel kan, of zullen we zelf een functie moeten schrijven die de plaats van atomen kan berekenen met de gegeven symmetrieoperaties. Zo een functie schrijven vergt veel werk, dit is dan ook de reden dat tot nu OpenBabel wordt gebruikt om dit te doen, maar is zeker mogelijk.

5.5.2 Bindingsafstand per atoom

Op dit moment gebruiken we één algemene variabele om te bepalen of atomen al dan niet gebonden moeten worden. Dit zouden we kunnen uitbreiden naar een dictionary die per atoom deze afstand bijhoudt. Op deze manier kan er een meer correct kristal worden getekend, en zal de kans op foute bindingen minder groot zijn. Doordat de bindingsafstand verschillend is per atoom zullen we in het huidige programma soms twee atomen binden die normaal geen binding zouden mogen hebben doordat ze binnen de gegeven afstand liggen. Om dit probleem op te lossen zou de gebruiker de bindingsafstand exact moeten kiezen zodat dit niet voorkomt, wat vaak zeer moeilijk of zelfs onmogelijk is. Een dictionary met de correcte bindingsafstand per atoom zou dit probleem oplossen. Figuur[5.5] toont links een voorbeeld waar ons programma bij de standaard bindingsafstand te veel bindingen maakt. Op de rechterafbeelding is de bindingsafstand verzet naar 1.75 en worden enkel de correcte bindingen getekend.

Een dictionary met bindingsafstanden zou op een gelijkaardige manier worden gemaakt als de dictionaries die de straal en de kleur van elementen bepalen. Deze zullen we in de code opgeroepen wanneer de afstand tussen twee atomen wordt vergeleken.



Figuur 5.5: Voorbeeld van kristal met foute bindingen (links) en correcte bindingen (rechts)

5.5.3 CCTBX

In de vierde sectie van hoofdstuk twee hebben we de *Computational Crystallography Toolbox* reeds

besproken als een alternatief voor PyCIFRW om CIF-bestanden te parsen. Naast het parsen van bestanden heeft de cctbx nog een hele reeks aan andere functionaliteiten, waarvan veel erg handig kunnen zijn voor ons programma. Zo zou de cctbx niet enkel OpenBabel kunnen vervangen om de plaats van de atomen binnen de eenheidscel te berekenen, het is zelfs in staat de positie van alle atomen binnen bepaalde grenzen te berekenen, zoals we eerder in deze sectie hebben besproken.

Het enige wat ons tegenhoudt cctbx te gebruiken is dat het erg moeilijk is deze werkende te krijgen op een Windowsssysteem. In tegenstelling tot de installatie op Linux, welke met slechts één commando kan gedaan worden, is de installatie op Windows erg lastig. Zowel de installatiehandleiding die te vinden is op de website van cctbx, als de handleiding op hun GitHub pagina is gedateerd en vooraleer de installatie kan worden uitgevoerd, moeten er een aantal programma's geïnstalleerd zijn. We willen ook zeker zijn dat het programma bruikbaar is op elk besturingssysteem en dat de installatie ervan niet te moeilijk is voor onervaren gebruikers. Dit alles heeft ertoe geleid dat er in dit onderzoek geen gebruik is gemaakt van de cctbx.

5.6 Conclusie

In dit hoofdstuk zagen we eerst een stappenplan voor het gebruiken van onze add-on. Dit dient als handleiding die gebruikers kunnen volgen om de add-on te gebruiken zonder enige voorkennis nodig te hebben.

Uit de resultaten die we uit de testen verkregen, kunnen we concluderen dat de tekenduur van de add-on voor een klein kristal in de grootteorde van enkele seconden is, terwijl dit voor grote kristallen naar enkele tientallen seconden neigt. De duur neemt ook toe wanneer we de tekenkwaliteit opdrijven. Het tekenen van bindingen zal aan deze tekenduur een vaste waarde toevoegen onafhankelijk van de tekenkwaliteit. Dit is omdat de kwaliteit enkel invloed heeft op het aantal segmenten van de atomen en niet die van de bindingen.

In vergelijking met VESTA duurt het tekenen van een kristal beduidend langer bij onze add-on. Bij een klein kristal duurt het ongeveer 17 maal langer en deze factor neemt toe tot wel 400 bij een groot kristal. Deze vertraging is voornamelijk te wijten aan de bpy module welke eerder traag is. Door het uitvoeren van enkele testen hebben we waargenomen dat gemiddeld 80 procent van de loopduur van ons programma naar het tekenen van de atomen gaat.

Op het vlak van gebruiksvriendelijkheid heeft onze add-on enkele verschillen ten opzichte van VESTA, zo heeft onze add-on minder features, maar is het eenvoudiger in gebruik en is het kristalmodel interactiever dan dat van VESTA. Welk programma nu juist het meest geschikt is op vlak van gebruiksvriendelijk hangt voornamelijk van de toepassing af. Een groot voordeel dat onze add-on heeft over VESTA is dat het enorm eenvoudig uit te breiden is, terwijl een gebruiker VESTA moeilijk of zelfs niet kan uitbreiden. Dit maakt de add-on erg "future proof".

Een probleem waar we in dit onderzoek mee hebben gekampert is het moeilijk vinden van online informatie over de nieuwste versie van de Blender API, die we gebruiken in onze add-on. Doordat de verschillen tussen Blender 2.80 en oudere versies van Blender zijn veel technieken die vroeger mogelijk waren niet meer van toepassing en zijn er weinig voorbeelden van toepassingen die wel werken. Door de goede documentatie van de Blender API was dit echter geen enorm probleem.

Door een probleem met de CifFile module is het tot heden niet mogelijk bestanden met lange padnamen in te lezen. Hierdoor kan onze add-on niet functioneren zonder dat de gebruiker OpenBabel heeft geïnstalleerd. Dit probleem kan voorlopig niet worden opgelost.

We hebben enkele features gezien die mogelijks kunnen worden toegevoegd aan de add-on. Door de gebruiker zelf grenzen te laten kiezen waartussen het kristal moet worden getekend kan deze zelf de grootte van het kristal bepalen, voorlopig worden enkel de atomen binnen de eenheidscel getekend.

Door een dictionary toe te voegen die de bindingsafstand per atoom bevat, hoeft de gebruiker niet meer manueel de bindingsafstand te bepalen. Dit zal leiden tot minder foute bindingen.

Ten slotte hebben we geleerd dat de cctbx een krachtige tool is met erg veel functies die in onze add-on kunnen worden gebruikt. Op dit moment wordt er in onze add-on nog geen gebruik gemaakt van de cctbx, dit komt doordat deze erg moeilijk te installeren is op een Windowssysteem.

Hoofdstuk 6

Besluit

In deze thesis hebben we onderzocht of het mogelijk is een interface te ontwerpen die het mogelijk maakt kristallen te visualiseren in het open source tekenprogramma Blender. Dit onderzoek hebben we onderverdeeld in drie algemene stappen: een studie van de *state of the art*, het ontwerpen van het programma en ten slotte het uitvoeren van testen.

In de eerste van deze stappen hebben we een stap gezet in de wereld van kristallografie, om zo een beter zicht te krijgen over hoe kristallen worden opgebouwd. Zo hebben we kennis gemaakt met de eenheidscel van een kristal. Dit is een driedimensionaal rooster waarbinnen de volledige structuur van een kristal zich bevindt. Ongeacht de grootte van een kristal, zal dit steeds bestaan uit een aaneenschakeling van deze eenheidscellen.

Een eenheidscel wordt steeds beschreven aan de hand van zes getallen die de roosterparameters worden genoemd. Drie van deze beschrijven de lengtes van de ribben, terwijl de overige drie de hoeken ertussen bepalen. De vorm van zo een eenheidscel in echter niet willekeurig, Bravais, een Franse natuurkundige, was in staat op basis van de vorm en symmetrie van kristallen 14 verschillende manieren gevonden waarmee kristalloosters kunnen worden opgebouwd, welke we de Bravaisroosters noemen.

Ook de posities van de atomen binnen een kristal zijn bepaald volgens enkele regels, dit wordt de inwendige symmetrie van een kristal genoemd en geeft ons een totaal van 230 verschillende manieren waarop deeltjes in een eenheidscel kunnen worden geordend.

Het CIF-formaat is een digitaal formaat waarin de data van een kristal wordt opgeslagen. Doordat dit formaat in ASCII-codering is geschreven, is dit leesbaar door zowel mens als computer. Dit maakt het CIF-formaat een erg geschikt formaat om als invoer te gebruiken voor onze interface. Een CIF-bestand bestaat steeds uit één of meerdere datablokken die een aantal parameters bevat. Deze parameters worden gevolgd door de waarde van deze parameters en bevatten alle informatie die nodig is om een kristal te tekenen. Het CIF-formaat geeft echter niet alle elementen die in de eenheidscel voorkomen maar beperkt zich tot de combinatie van elementen en inwendige symmetrieoperaties waarmee de positie van elk element kan berekend worden. Voor deze berekening hebben we gebruik gemaakt van OpenBabel. OpenBabel maakt met de gegeven data een nieuw CIF-bestand aan dat exact hetzelfde kristal beschrijft, maar waar in plaats van symmetrieoperaties,

de lijst van alle atomen in de eenheidscel wordt gegeven.

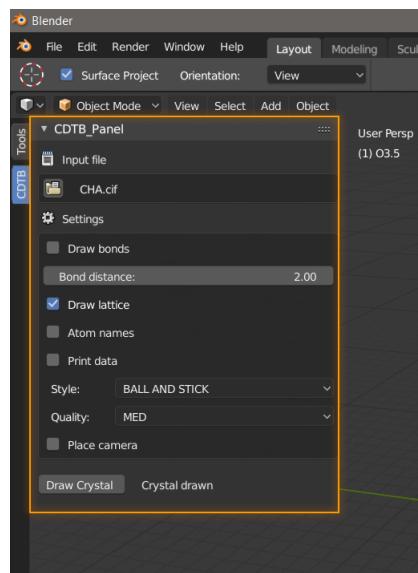
Vooraleer we dit CIF-bestand konden visualiseren moesten we op zoek gegaan naar een manier om de data uit het CIF-formaat in meer bruikbare datastructuren op te slaan. Dit hebben we gedaan door het CIF-bestand te parsen met de CifFile module van PyCIFRW en de verkregen data op te slaan in verschillende klassen die we hebben aangemaakt.

Om objecten te tekenen in Blender moesten we gebruik maken van de bpy module van de Blender API. Naast het tekenen van atomen wilden we ook de mogelijkheid hebben tot het tekenen van een omkadering van de eenheidscel en eventuele bindingen tussen atomen. Eerst zal de omkadering getekend worden door bollen te tekenen op de hoekpunten van de cel en tussen deze cilinders te tekenen.

Voor het tekenen van de atomen hebben we twee externe dictionaries gemaakt die voor elk bestaand element de straal en de kleur van het atoom bijhouden. Vervolgens wordt de lijst met atomen doorlopen en bepalen we voor elk atoom de positie, straal en kleur. Ten slotte berekenen we de afstand tussen alle atomen, en tekenen we een binding tussen twee atomen wanneer hun onderlinge afstand minder is dan een door de gebruiker bepaalde waarde.

Ten slotte hebben we gekozen een Blender add-on te maken van ons programma, waardoor het eenvoudig kan gebruikt worden in de Blender omgeving. Door het programma als een add-on aan te bieden kan een gebruiker met behulp van een grafische interface kristallen tekenen in plaats van het programma manueel uit te voeren in Blender.

Het eerste deel van het ontwerpen van de add-on, te zien op Figuur[6.2], was het maken van operatoren die de functionaliteiten van het programma aansturen. Het tweede deel van het ontwerpen van de add-on was het ontwerpen van een paneel. Met een paneel kan de opmaak en structuur van de add-on gepersonaliseerd worden.



Figuur 6.1: De add-on zoals te zien in Blender

We hebben ook enkele testen uitgevoerd en nadien de resultaten vergeleken met het kristalvisualisatieprogramma VESTA. Op vlak van snelheid deed ons programma er ongeveer een halve seconde over om een klein kristal met bindingen te tekenen, terwijl dit voor een groot kristal meer dan twee minuten duurde. Het tekenen van het kleine kristal deed VESTA 17 maal sneller en het grote kristal tekende VESTA zelfs 400 maal sneller dan ons programma. Op vlak van gebruiksvriendelijkheid hing het vooral af van de toepassing, hoewel VESTA dankzij het grote aantal aan features het meest geschikt was voor wetenschappelijk gebruik, was het minder eenvoudig in omgang en was het kristalmodel minder interactief.

Ons programma overtroefte VESTA wel op vlak van uitbreidbaarheid. Dit was omdat het niet mogelijk is als gebruiker features toe te voegen aan VESTA, terwijl ons programma zo geschreven is dat het toevoegen van features relatief eenvoudig is.

Een nadeel van ons programma is dat het gemaakt is voor de allernieuwste versie van Blender, die eigenlijk nog in beta is. Hierdoor is het moeilijker informatie te vinden over de functionaliteiten van de Blender API, ondanks de duidelijke documentatie.

Er is een functionaliteit in ons programma die niet werkt door een probleem in de CifFile module van PyCIFRW. Dit probleem leidt ertoe dat er geen lange padnamen kunnen worden meegegeven met de inleesfunctie waardoor enkel bestanden in de werkmap kunnen worden ingelezen. Hierdoor is er steeds nood aan OpenBabel, terwijl het programma ook zonder zou moeten kunnen lopen.

Door de uitbreidbaarheid van het programma en de mogelijkheden die de Blender API biedt, is *the sky the limit*. Enkele features die zeker interessant zijn in de toekomst toe te voegen zijn:

- De gebruiker de mogelijkheid geven de grenzen te bepalen waarbinnen getekend moet worden
- Een individuele bindingsafstand per element zodat foute bindingen vermeden kunnen worden
- Het implementeren van modules uit de cctbx

Bibliografie

- Bernstein, H. J., Bollinger, J. C., Brown, I. D., Gražulis, S., Hester, J. R., McMahon, B., Spadaccini, N., Westbrook, J. D., and Westrip, S. P. (2016). Specification of the crystallographic information file format, version 2.0. *Journal of Applied Crystallography*, 49(1):277–284.
- Blender (2019). Blender2.80 python api documentation. <https://docs.blender.org/api/blender2.8/>.
- Blenderguru (2012). Photorealism competition results. <https://www.blenderguru.com/competitions/photorealism-competition-results>.
- Borchardt-Ott, W. (2011). *Crystallography: an introduction*. Springer Science and Business Media.
- Britannica (2018). Auguste bravais: French physicist. <https://www.britannica.com/biography/Auguste-Bravais>.
- Chronister, J. (2011). *Blender Basics: Classroom Tutorial Book*. Blender Nation.
- Conlan, C. (2017). *The Blender Python API: Precision 3D Modeling and Add-on Development*. Apress.
- Hall, S. R., Allen, F. H., and Brown, I. D. (1991). The crystallographic information file (cif): a new standard archive file for crystallography. *Acta Crystallographica Section A: Foundations of Crystallography*, 47(6):655–685.
- IUCR (2009). Programming with pycifrw and pyxtarw. https://www.iucr.org/__data/iucr/cif/software/pycifrw/PyCifRW-3.2/documentation.pdf.
- IZA (2018). Database of zeolite structures. http://europe.iza-structure.org/IZA-SC/ftc_table.php.
- Mike (2018). Python 3: An intro to enumerations. <https://www.blog.pythonlibrary.org/2018/03/20/python-3-an-intro-to-enumerations/>.
- Momma, K. and Izumi, F. (2008). Vesta: a three-dimensional visualization system for electronic and structural analysis. *Journal of Applied Crystallography*, 41(3):653–658.
- OpenBabel (2016). Open babel: The open source chemistry toolbox. http://openbabel.org/wiki/Main_Page. GitHub: <https://github.com/openbabel/openbabel>.

Tangent (2018). Next gen. <https://www.tangent-animation.com/next-gen>.

Theo Hahn, H. W. and Muller, U. (2005). *International Tables for Crystallography Volume A: Space-Group Symmetry*. by SPRINGER for THE INTERNATIONAL UNION OF CRYSTALLOGRAPHY.

Tregonning, P. (2007). Bezier curves animated quintic bezier curve. https://en.wikipedia.org/wiki/B%C3%A9zier_curve.

Bijlagen

Bijlage A (**op SD**) De CIF-dictionary

Bijlage B: Cif-bestand van de kristalstructuur van Chaziet

Bijlage C (**op SD**): Zelf ontworpen CIF-parser

Bijlage D (**op SD**): Volledige code van het programma

Bijlage E : Externe dictionary met stralen van elementen

Bijlage F : Externe dictionary met kleuren van elementen

Bijlage G : Specs van het gebruikte systeem

Bijlage H (**op SD**): ZIP bestand met daarin de add-on

Bijlage B

```

1  data_CHA
2  #*****#
3  #
4  # CIF taken from the IZA-SC Database of Zeolite Structures
5  # Ch. Baerlocher and L.B. McCusker
6  # Database of Zeolite Structures: http://www.iza-structure.org/databases/
7  #
8  # The atom coordinates and the cell parameters were optimized with DLS76
9  # assuming a pure SiO2 composition.
10 #
11 #*****#
12
13 _cell_length_a          13.6750(0)
14 _cell_length_b          13.6750(0)
15 _cell_length_c          14.7670(0)
16 _cell_angle_alpha       90.0000(0)
17 _cell_angle_beta        90.0000(0)
18 _cell_angle_gamma       120.0000(0)
19
20 _symmetry_space_group_name_H-M      'R -3 m'
21 _symmetry_Int_Tables_number         166
22 _symmetry_cell_setting            trigonal
23
24 loop_
25 _symmetry_equiv_pos_as_xyz
26 '+x,+y,+z'
27 '2/3+x,1/3+y,1/3+z'
28 '1/3+x,2/3+y,2/3+z'
29 '-y,+x-y,+z'
30 '2/3-y,1/3+x-y,1/3+z'
31 '1/3-y,2/3+x-y,2/3+z'
32 '-x+y,-x,+z'
33 '2/3-x+y,1/3-x,1/3+z'
34 '1/3-x+y,2/3-x,2/3+z'
35 '-y,-x,+z'
36 '2/3-y,1/3-x,1/3+z'
37 '1/3-y,2/3-x,2/3+z'
38 '-x+y,+y,+z'
39 '2/3-x+y,1/3+y,1/3+z'
40 '1/3-x+y,2/3+y,2/3+z'
41 '+x,+x-y,+z'
42 '2/3+x,1/3+x-y,1/3+z'
43 '1/3+x,2/3+x-y,2/3+z'
44 '-x,-y,-z'
45 '2/3-x,1/3-y,1/3-z'
46 '1/3-x,2/3-y,2/3-z'
47 '+y,-x+y,-z'
48 '2/3+y,1/3-x+y,1/3-z'
49 '1/3+y,2/3-x+y,2/3-z'
50 '+x-y,+x,-z'
51 '2/3+x-y,1/3+x,1/3-z'
52 '1/3+x-y,2/3+x,2/3-z'
53 '+y,+x,-z'
54 '2/3+y,1/3+x,1/3-z'
55 '1/3+y,2/3+x,2/3-z'
56 '+x-y,-y,-z'
57 '2/3+x-y,1/3-y,1/3-z'
58 '1/3+x-y,2/3-y,2/3-z'
59 '-x,-x+y,-z'
60 '2/3-x,1/3-x+y,1/3-z'
61 '1/3-x,2/3-x+y,2/3-z'
62
63 loop_
64 _atom_site_label
65 _atom_site_type_symbol
66 _atom_site_fract_x
67 _atom_site_fract_y
68 _atom_site_fract_z
69     O1    O    0.9020    0.0980    0.1227
70     O2    O    0.9767    0.3101    0.1667
71     O3    O    0.1203    0.2405    0.1315
72     O4    O    0.0000    0.2577    0.0000
73     T1    Si   0.9997    0.2264    0.1051

```

Bijlage E

```
1 {  
2     "H"      : .25,  
3     "He"     : 1.20,  
4     "Li"     : 1.45,  
5     "Be"     : 1.05,  
6     "B"      : .85,  
7     "C"      : .7,  
8     "N"      : .65,  
9     "O"      : .6,  
10    "F"      : .5,  
11    "Ne"     : 1.60,  
12    "Na"     : 1.8,  
13    "Mg"     : 1.5,  
14    "Al"     : 1.25,  
15    "Si"     : 1.1,  
16    "P"      : 1,  
17    "S"      : 1,  
18    "Cl"     : 1,  
19    "Ar"     : .71,  
20    "K"      : 2.2,  
21    "Ca"     : 1.8,  
22    "Sc"     : 1.6,  
23    "Ti"     : 1.4,  
24    "V"      : 1.35,  
25    "Cr"     : 1.4,  
26    "Mn"     : 1.4,  
27    "Fe"     : 1.4,  
28    "Co"     : 1.35,  
29    "Cu"     : 1.35,  
30    "Ni"     : 1.35,  
31    "Zn"     : 1.35,  
32    "Ga"     : 1.3,  
33    "Ge"     : 1.25,  
34    "As"     : 1.15,  
35    "Se"     : 1.15,  
36    "Br"     : 1.15,  
37    "Kr"     : 1,  
38    "Rb"     : 2.35,  
39    "Sr"     : 2,  
40    "Y"      : 1.8,  
41    "Zr"     : 1.55,  
42    "Nb"     : 1.45,  
43    "Mo"     : 1.45,  
44    "Tc"     : 1.35,  
45    "Ru"     : 1.3,  
46    "Rh"     : 1.35,  
47    "Pd"     : 1.40,  
48    "Ag"     : 1.6,  
49    "Cd"     : 1.55,  
50    "In"     : 1.55,  
51    "Sn"     : 1.45,  
52    "Sb"     : 1.45,  
53    "Te"     : 1.4,  
54    "I"      : 1.4,  
55    "Xe"     : 2,  
56    "Cs"     : 2.6,  
57    "Ba"     : 2.15,  
58    "La"     : 1.95,  
59    "Ce"     : 1.85,  
60    "Pr"     : 1.85,  
61    "Nd"     : 1.85,  
62    "Pm"     : 1.85,  
63    "Sm"     : 1.85,  
64    "Eu"     : 1.85,  
65    "Gd"     : 1.8,  
66    "Tb"     : 1.75,  
67    "Dy"     : 1.75,  
68    "Ho"     : 1.75,  
69    "Er"     : 1.75,  
70    "Tm"     : 1.75,  
71    "Yb"     : 1.75,  
72    "Lu"     : 1.75,  
73    "Hf"     : 1.55,
```

```
74      "Ta"      :      1.45,
75      "W"       :      1.35,
76      "Re"      :      1.35,
77      "Os"      :      1.3,
78      "Ir"      :      1.35,
79      "Pt"      :      1.35,
80      "Au"      :      1.35,
81      "Hg"      :      1.5,
82      "Tl"      :      1.9,
83      "Pb"      :      1.8,
84      "Bi"      :      1.6,
85      "Po"      :      1.9,
86      "Ra"      :      2.15,
87      "Ac"      :      1.95,
88      "Th"      :      1.8,
89      "U"       :      1.75,
90      "Np"      :      1.75,
91      "Pu"      :      1.75,
92      "Am"      :      1.75,
93  }
94
```

Bijlage F

```

1   {
2     "H"      : [1,1,1],
3     "He"     : [0,1,1],
4     "Li"     : [1,0,1],
5     "Be"     : [0,0.5,0],
6     "B"      : [1,0.7,0.7],
7     "C"      : [0,0,0],
8     "N"      : [0,0,1],
9     "O"      : [1,0,0],
10    "F"      : [0,1,0],
11    "Ne"     : [0,1,1],
12    "Na"     : [1,0,1],
13    "Mg"     : [0,0.5,0],
14    "Al"     : [1,0.4,0.7],
15    "Si"     : [1,0.7,0.7],
16    "P"      : [1,0.5,0],
17    "S"      : [1,1,0],
18    "Cl"     : [0,1,0],
19    "Ar"     : [0,1,1],
20    "K"      : [1,0,1],
21    "Ca"     : [0,0.5,0],
22    "Sc"     : [1,0.4,0.7],
23    "Ti"     : [0.5,0.5,0.5],
24    "V"      : [1,0.4,0.7],
25    "Cr"     : [1,0.4,0.7],
26    "Mn"     : [1,0.4,0.7],
27    "Fe"     : [1,0.3,0],
28    "Co"     : [1,0.4,0.7],
29    "Ni"     : [1,0.4,0.7],
30    "Cu"     : [1,0.4,0.7],
31    "Zn"     : [1,0.4,0.7],
32    "Ga"     : [1,0.4,0.7],
33    "Ge"     : [1,0.4,0.7],
34    "As"     : [1,0.4,0.7],
35    "Se"     : [1,0.4,0.7],
36    "Br"     : [0.4,0,0],
37    "Kr"     : [0,1,1],
38    "Rb"     : [1,0,1],
39    "Sr"     : [0,0.5,0],
40    "Y"      : [1,0.4,0.7],
41    "Zr"     : [1,0.4,0.7],
42    "Nb"     : [1,0.4,0.7],
43    "Mo"     : [1,0.4,0.7],
44    "Tc"     : [1,0.4,0.7],
45    "Ru"     : [1,0.4,0.7],
46    "Rh"     : [1,0.4,0.7],
47    "Pd"     : [1,0.4,0.7],
48    "Ag"     : [1,0.7,0.7],
49    "Cd"     : [1,0.4,0.7],
50    "In"     : [1,0.4,0.7],
51    "Sn"     : [1,0.4,0.7],
52    "Sb"     : [1,0.4,0.7],
53    "Te"     : [1,0.4,0.7],
54    "I"      : [0.3,0,0.3],
55    "Xe"     : [0,1,1],
56    "Cs"     : [1,0,1],
57    "Ba"     : [0,0.5,0],
58    "La"     : [1,0.4,0.7],
59    "Ce"     : [1,0.4,0.7],
60    "Pr"     : [1,0.4,0.7],
61    "Nd"     : [1,0.4,0.7],
62    "Pm"     : [1,0.4,0.7],
63    "Sm"     : [1,0.4,0.7],
64    "Eu"     : [1,0.4,0.7],
65    "Gd"     : [1,0.4,0.7],
66    "Tb"     : [1,0.4,0.7],
67    "Dy"     : [1,0.4,0.7],
68    "Ho"     : [1,0.4,0.7],
69    "Er"     : [1,0.4,0.7],
70    "Tm"     : [1,0.4,0.7],
71    "Yb"     : [1,0.4,0.7],
72    "Lu"     : [1,0.4,0.7],
73    "Hf"     : [1,0.4,0.7],

```

```
74      "Ta"      :      [1,0.4,0.7],  
75      "W"       :      [1,0.4,0.7],  
76      "Re"      :      [1,0.4,0.7],  
77      "Os"      :      [1,0.4,0.7],  
78      "Ir"      :      [1,0.4,0.7],  
79      "Pt"      :      [1,0.4,0.7],  
80      "Au"      :      [1,0.7,0.7],  
81      "Hg"      :      [1,0.4,0.7],  
82      "Tl"      :      [1,0.4,0.7],  
83      "Pb"      :      [1,0.4,0.7],  
84      "Bi"      :      [1,0.4,0.7],  
85      "Po"      :      [1,0.4,0.7],  
86      "Ra"      :      [0,0.5,0],  
87      "Ac"      :      [1,0.4,0.7],  
88      "Th"      :      [1,0.4,0.7],  
89      "U"       :      [1,0.4,0.7],  
90      "Np"      :      [1,0.4,0.7],  
91      "Pu"      :      [1,0.4,0.7],  
92      "Am"      :      [1,0.4,0.7],  
93  }  
94
```

Bijlage G

Computer specs.

Processor	
Processor	Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz
Manufacturer	Intel
Speed	3.1 GHz
Number of Cores	4
CPU ID	BFEFBFF000806E9
Family	06
Model	8E
Stepping	9
Revision	
Memory	
RAM	8.0 GB
Video Card	
Video Card	Intel(R) HD Graphics 620
Manufacturer	
Chipset	Intel(R) HD Graphics 620
Dedicated Memory	128 MB
Total Memory	5.0 GB
Pixel Shader Version	
Vertex Shader Version	
Hardware T & L	Yes
Vendor ID	8086
Device	5916
Plug and Play ID	VEN_8086&DEV_5916&SUBSYS_827E103C&REV_02
Driver Version	23.20.16.4973
Operating System	
Operating System	Windows 10
Service Pack	0
Size	64-bit
Edition	
Version	10.0.17134
Locale	0809

FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN
CAMPUS DE NAYER SINT-KATELIJNE-WAVER
J. De Nayerlaan 5
2860 SINT-KATELIJNE-WAVER, België
tel. + 32 15 31 69 44
iw.denayer@kuleuven.be
www.iw.kuleuven.be

