KU LEUVEN

FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN

CAMPUS DE NAYER SINT-KATELIJNE-WAVER

Wandelen door kristallen

Blender als kristallografische visualisatietool

Jarrit BOONS

Promotor: Ann Phillips

Co-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

Academiejaar 2018 - 2019

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

Lijst van afkortingen en begrippen

ASCII
American Standard Code for Information Interchange, standaard voor de codering van karakters
API
Application Programming Interface, een verzameling van definities en methodes
Open source
Een product dat vrij mag gebruikt worden
GUI
Graphic User Interface, een visuele interface tussen een programma en de gebruiker
Interface
Manier van interactie tussen twee zaken
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 onderzoeksdomeinen 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 beknopte 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 langdradige 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 voornoemde 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 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 mogelijk 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 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 Onderzoeksvragen

De doelstellingen uit vorige sectie kunnen worden geformuleerd als onderstaande onderzoeksvragen.
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 kunnen 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 heeft als doel 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 gedachtegang 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 onderzoeksvragen 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 kirstalstructuren 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.

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

Tabel 2.1 Kristallen in wetenschappelijke takken

2.1 Kristallografie

2.1.1 Wat is kristallografie

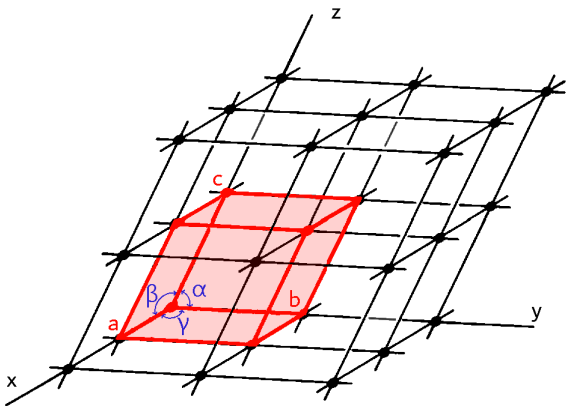
Kristallografie kan beschreven worden als de studie van materie in een kristallijne staat en houdt zich onder andere bezig met de synthese en opbouw van kristallen en hun fysische en chemische eigenschappen. Eerder in dit hoofdstuk werd kristallografie beschreven als een specifieke tak van de wetenschap, 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 parallellogrammen 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.

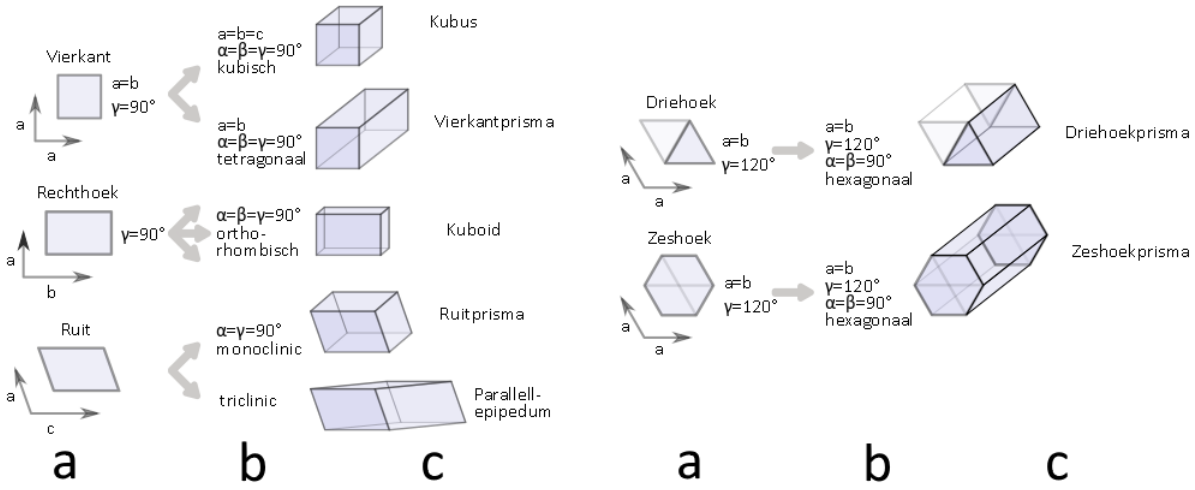


Figuur 2.1: Voorstelling van een eenheidscel in de roosterruimte (Borchardt-Ott, 2011)

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, orthorhombisch, monoclinisch, triclinisch, en hexagonaal. Deze worden weergegeven in kolom b van figuur[2.2].

Vertrekkende 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

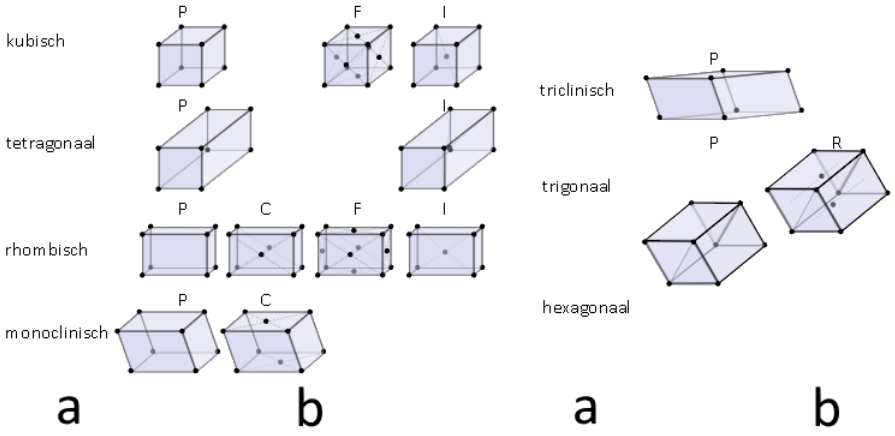
De Franse natuurkundige Bravais (Britannica, 2018) geeft een extra indeling op basis van mogelijke translaties binnen de eenheidscel, welke in overeenkomst met de symmetrievoorwaarden van de kristalsystemen zijn. Dit leidt tot de 14 Bravais roosters, ook wel gecentreerde roosters genoemd welke worden 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 symmetrievoorwaarden 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.

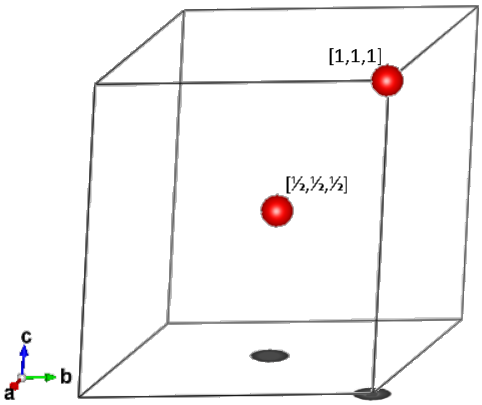
2.1.5 Beschrijving van een kristal

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.

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

Met Ω het volume van de eenheidscel.

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

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

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

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 datawerking 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.

```
#                                                           1
    **************************************************************************

# CIF taken from the IZA-SC Database of Zeolite Structures  2
# Ch. Baerlocher and L.B. McCusker                          3
#                                                           4
    **************************************************************************
```

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 naar 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

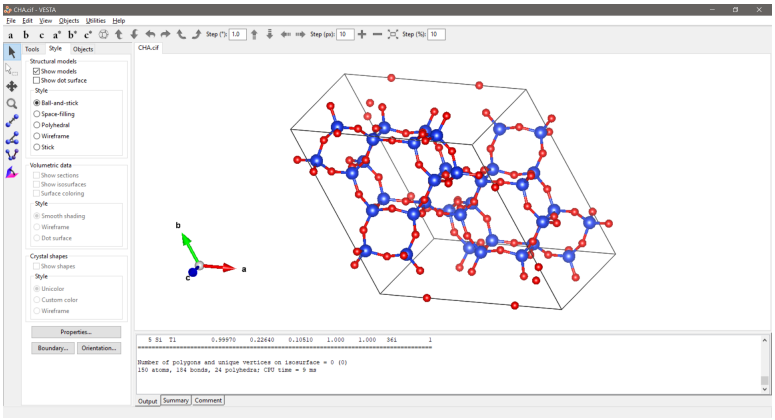
Aangezien het CIF-formaat gezien kan worden als een standaardformaat voor het beschrijven van kristallen bestaan er reeds CIF-bestanden voor vrijwel alle bestaande kristalstructuren. De ver-

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 kirstalstructuren 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 zeoliettype Chabaziet, dit CIF-bestand kan teruggevonden worden in de zeolietstructuurdatabank 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 zeoliet.

2.3 Kristalvisualisatiesoftware

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, transleren 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 zijn 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.

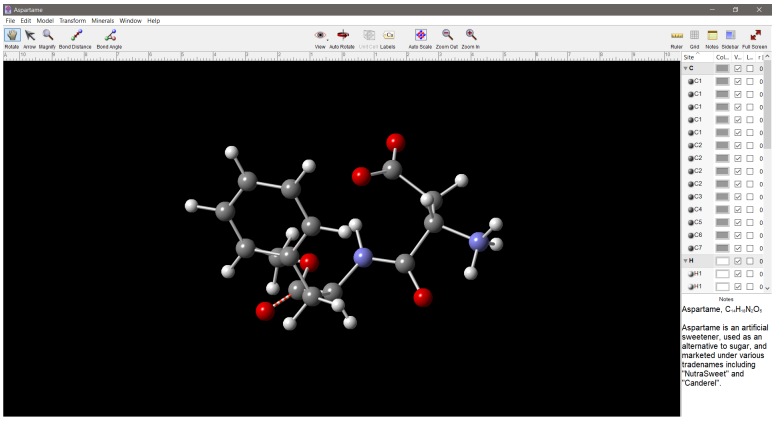


Figuur 2.5: De user interface van VESTA3

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

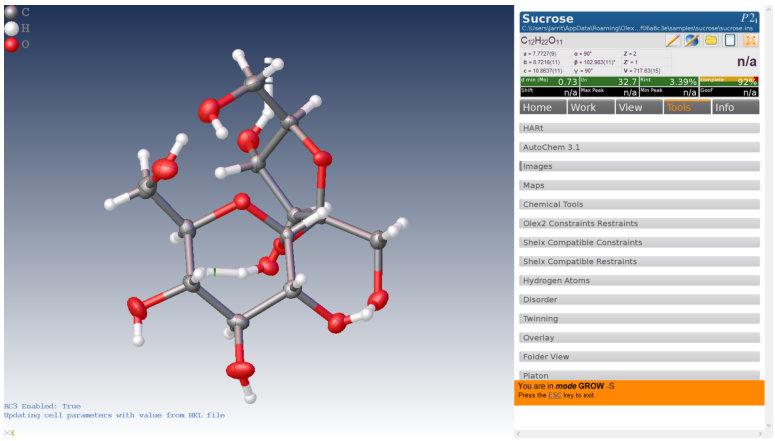
Naast VESTA bestaat er nog een groot aanbod aan andere software die in staat zijn kristallen driedimensionaal te visualiseren. Het Crystalmaker softwarepakket biedt het programma Crystalviewer, figuur[2.6], gratis aan voor persoonlijk gebruik. Het strak design en de gebruiksvriendelijke user interface 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 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 Olex^2, zie figuur[2.7]. Olex^2 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. Olex^2 is volledig geschreven in Python en maakt gebruik van enkele bibliotheken die de gebruiker de mogelijkheid biedt verschillende kristalbeschrijvingsformaten, waaronder CIF, in te lezen te visualiseren. Ondanks de features en documentatie is het gebruiken van Olex^2 minder eenvoudig dan de eerdere geziene 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 Olex^2

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 Olex^2, 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"]
#De data uit een lusblok van het datablok kan worden opgevraagd met
lb = datablok.GetLoop("een datanaam uit de loop")
```

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.

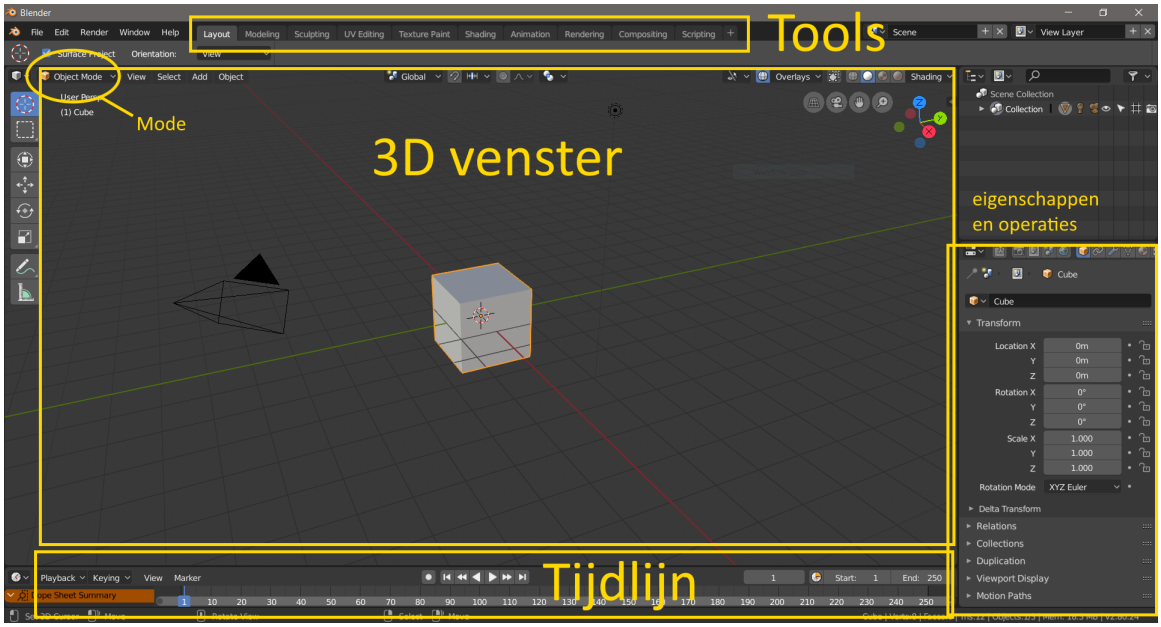
2.5.2 Blender in de bedrijfswereld

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 uitsluiten 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)



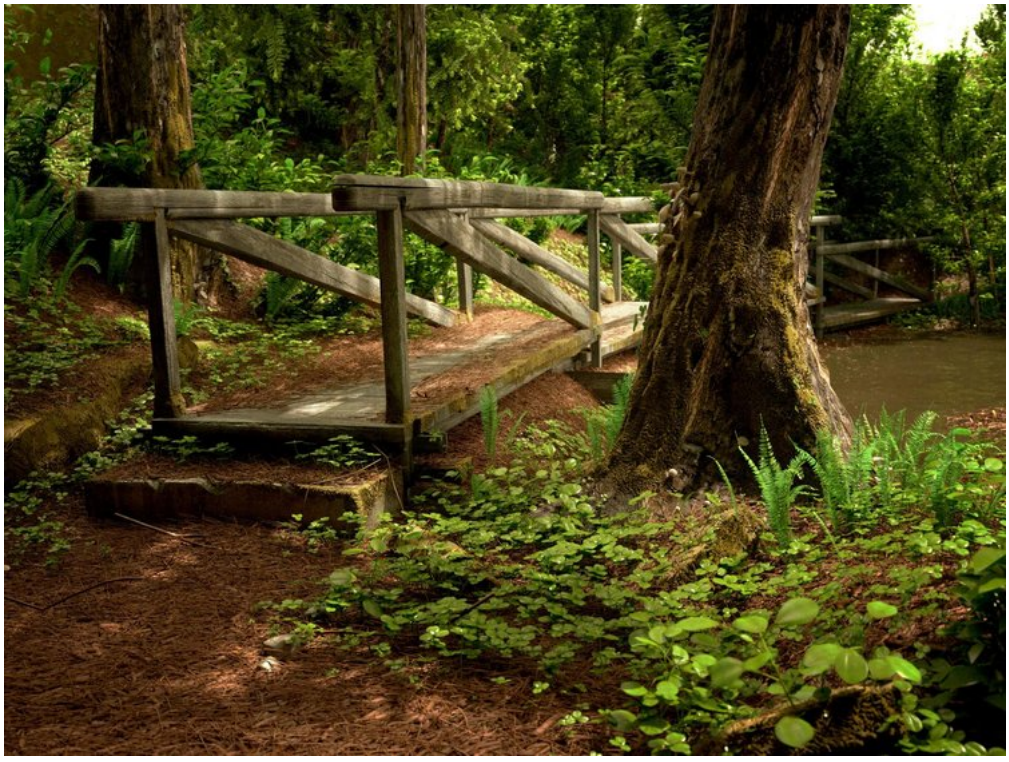
Figuur 2.9: Het Blender startscherm

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 modes: 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 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 levensecht te laten lijken. (Blenderguru, 2012)



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

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

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

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

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 in 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 creë ren die automatisch 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.

2.6 Conclusie

Kristallografie is een speciefieke 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 kristalvisualisatiesoftwarepakketten zijn vaak moeilijk in omgang, duur of bieden niet genoeg vrijheid. VESTA en Olex^2 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

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, hoe, met behulp van een parser, een CIF-bestand kan worden omgezet in verwerkbare data. Ten slotte wordt vertelt 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 vijfde 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.

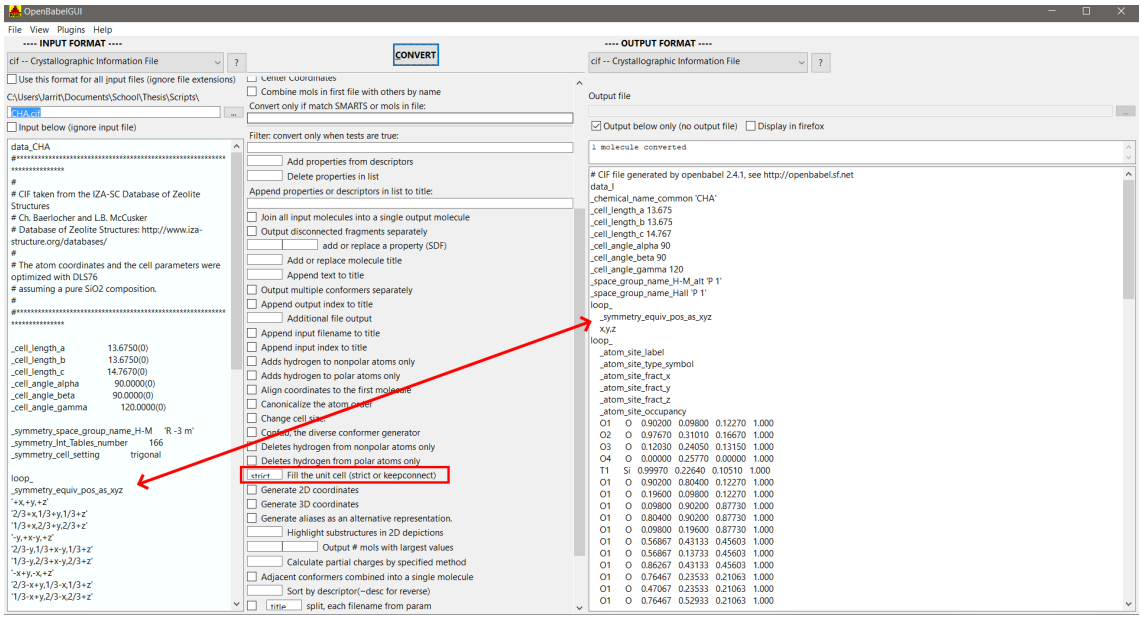
Aan de hand van de ruimtegroep en de centering van het kristalrooster kunnen alle symmetrieope-

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 symmetrieoperaties 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 meermaals 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)



Figuur 3.1: Conversie van ruimtegroep met OpenBabel GUI

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.

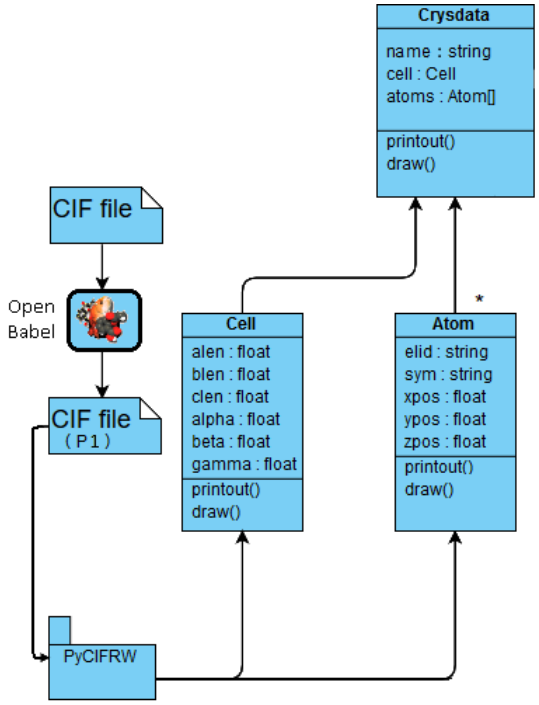
Hoewel de OpenBabel GUI het converteren van data erg eenvoudig en overzichtelijk maakt, zou het niet erg efficiënt zijn dat dit proces manueel moet worden gedaan voor elk kristal. Deze software kan echter ook vanuit een terminal worden uitgevoerd, wat, in combinatie met de subprocess module van python,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

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 object georiënteerd 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 Tekenen 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 con-

versiematrix 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 *Crysdata*. 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

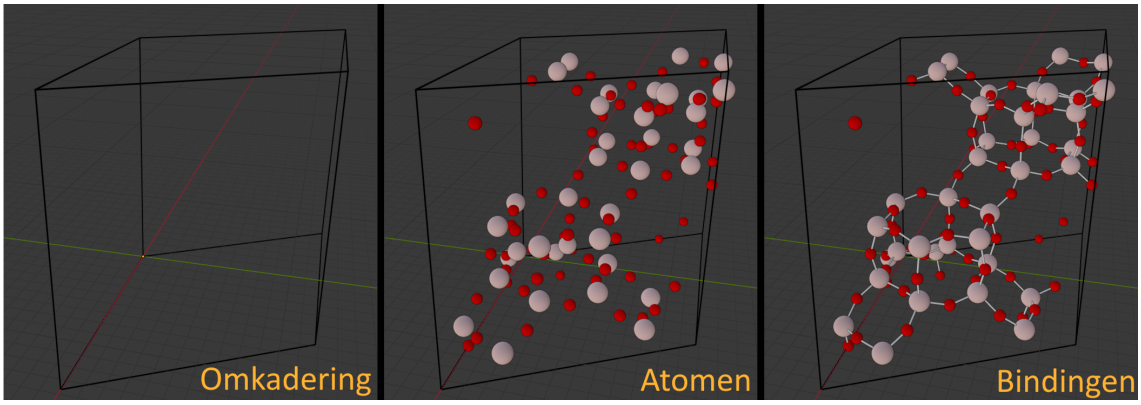
Het programma laat ook toe bindingen tussen atomen te tekenen, op basis van hun onderlinge afstand. De gebruiker kan een variabele aanpassen die bepaalt hoe groot de afstand tussen twee atomen maximaal mag zijn, zodat de binding tussen deze gemaakt zal worden.

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 gebruker 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 de 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.

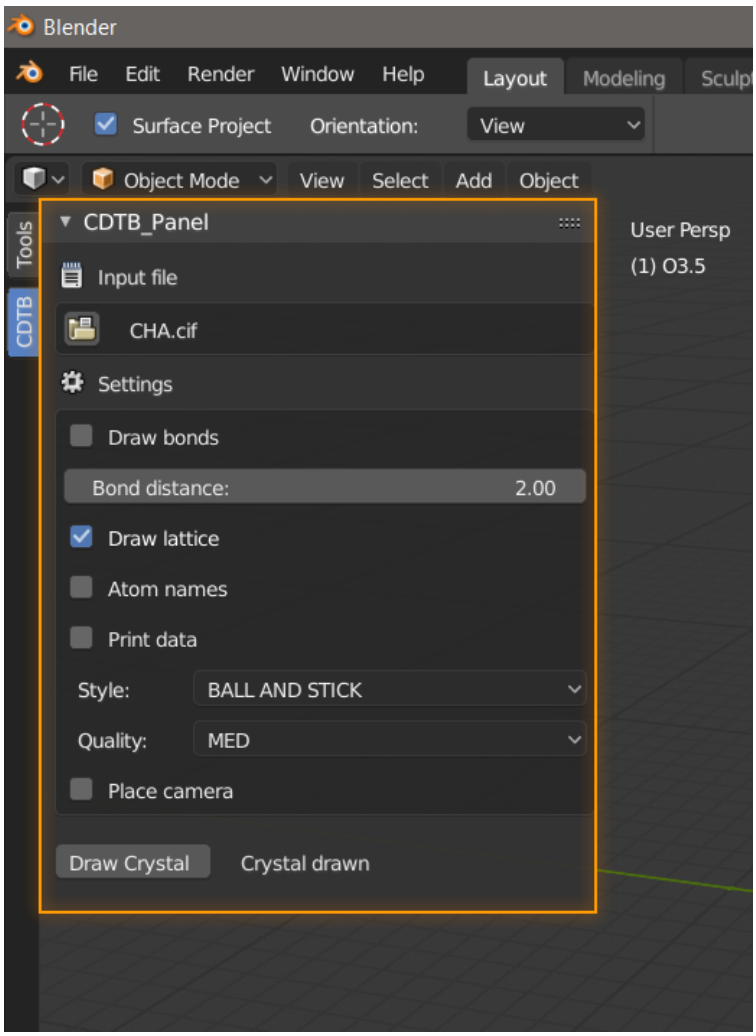
In het programma worden twee operators aangemaakt. Eén operator heeft als functie een bestandsbrowser te openen waarmee de gebruiker het CIF-bestand kan selecteren dat moet worden gevisualiseerd. De tweede operator voert het hele script uit dat in dit hoofdstuk beschreven wordt, vanaf het converteren 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. Booleans 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 kennisgemaakt 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

Werking van het programma

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 gedachtegang 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 bijlages 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.

4.1 Installatie van externe programma's

Zoals in hoodstuk 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 Windowssytemen 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 installatiewizard 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

Voor de PyCIFRW module op ons systeem te installeren hebben we nood aan een prompt waarop Pyhton3.7 kan worden uitgevoerd. Zo'n prompt kan verkregen worden door Anaconda te installeren en gebruik te maken van *Anaconda prompt*. Vervolgens hebben we de pip installer nodig voor Python en 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 het volgende lijn code uit te voeren:

```
import CifFile
```

Als er geen foutmelding wordt gegeven wilt het zeggen dat de module correct is geïnstalleerd op voor 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 in de installatiemap van Anaconda in de submap *site-packages* (Anaconda3 $>>$ Lib $>>$ site-packages). Vervolgens moeten we de 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 dergelijk een CIF-bestand is, dit zou anders voor problemen kunnen zorgen in het verdere verloop van het programma.

Om dit te controleren moeten we nazien 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 725 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 foutboodschap weergeven in de terminal en gaan we in de *user_feedback* variabele ook een boodschap 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:]                                    730
if(ext.lower() != ".cif"):                                  731
    print("Only cif files can be visualised")               732
    user_feedback = "Not a cif file"                        733
    return                                                  734
```

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 foutboodschap 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 zelfs geen inwendige symmetrie, en heeft de gebruiker 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-bestan 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 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'])
```

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 me 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 omdat deze allemaal dezelfde tekststructuur hebben. Bestanden uit databanken die werken met een andere structuur kunnen verkeerd worden geïnterpreteerd waardoor de parser nutteloos is. Het is mogelijk mijn parser aan te passen zodat, ongeacht de structuur van het CIF-bestand, een juiste interpretatie van de data kan gedaan worden. Dit zou echter veel tijd vergen en zal, gezien er reeds werkende CIF-parsers bestaan, niet worden gedaan.

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:                                                              16
    from CifFile import CifFile                                   17
    pars_check = False                                            18
except:                                                           19
    print("PyCIFRW not installed, try: pip install PyCifRW")      20
    pars_check = True                                             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 insteressante 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].

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

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 lusblok met atom in *CifFile* object te doorlopen en deze in een lijst plaatsen. Dit wordt gedaan in Listing[4.7]

```
639  def readEl(cb):
640
641      elements = []
642      previd = []
643      idcnt = []
644      lb = cb.GetLoop("_atom_site_label")
645      for el in lb:
646          flag = False
647          for i in range(len(previd)):
648              if(el[0] == previd[i]):
649                  flag = True
650                  break
651          if(flag):
652              idcnt[i] += 1
```

```
        else:                                                          653
            previd.append(el[0])                                       654
            idcnt.append(0)                                            655
            i = len(idcnt)-1                                           656
        id_t = "{}.{}".format(el[0],idcnt[i])                          657
        elements.append(Atom(id_t,el[1],el[2],el[3],el[4]))            658
    return elements                                                    659
```

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 gebrukt 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, die 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 foutboodschap 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 de gebruker 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 in als attributen van het *cell* object, eerst

kleine bollen tekenen op de hoekpunten van de eenheidscel. Dit wordt gedaan in Listing[4.8].

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

Listing 4.8: Tekenen 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 textitprimitive_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 gebruken 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 hoekpunten 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 specifiëren tussen dewelke we een lijn willen tekenen. We gaan twee even lange lijsten aanmaken waarin de hoekpunten staan die we moeten 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 een paar van objecten uit de lijst 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")
for i in cell_edges:
    i.select_set(action="SELECT")
# set corner in origin as active and join meshes as one object
bpy.context.view_layer.objects.active = cell_corners[0]
bpy.ops.object.join()
```

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 Tekenen 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):
    size = sizedic[self.sym]*styledic[draw_style][0]+bond_radius*styledic[draw_style][1]
    bpy.ops.mesh.primitive_uv_sphere_add(segments=qualitydic[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
    activeObject = bpy.context.active_object # Set active object to variable
    mat = bpy.data.materials.new(name="MaterialName") # set new material to variable
    activeObject.data.materials.append(mat) # add the material to the object
    if(atom_name):
        bpy.context.object.show_name = True
    if(atom_color):
        bpy.context.object.active_material.diffuse_color = colordic[self.sym] # change color to dictiomary color
```

```
        else:                                                          620
            bpy.context.object.active_material.diffuse_color =         621
                [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 gebuiker. Door het grootte aantal elementen zijn het te veel lijnen om in het programma zelf te schrijven. Om deze reden gaan we deze dictionaries opslaan in een extern tekstbestand. 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())                                        89
```

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 weten wordt de conversiematrix gebruikt. Deze zal de fractionele coördinaten, die atributen zijn van het atoom, 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 en bij de *STICK* tekenstijl zal de straal zo klein zijn dat enkel de bindingen nog zichtbaar zijn. De arumenten *segments* en *ring_count* op lijn 611 van Listing[4.10] bepalen uit hoeveel vlakken het de bol bestaat, bij een lage waarde zal de bol eerder hoekig zijn. Het aantal van deze segmenten kan de gebruiken kiezen en wordt aan de hand van een dictionary ingevuld.

Op lijn 612 van Listing[4.10] kennen we de getekende bol een naam toe, via deze naam kunnen we onze bol later terug terugvinden in de lijst van getekende objecten. Hierna gaan we onze bol terug 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 Tekenen van bindingen

In de *BALL AND STICK* tekenstijl heeft de gebruiker de mogelijkheid om bindingen tussen atomen te tekenen, in de STICK tekenstijl zullen de bindingen steeds worden getekend terwijl deze nooit zullen getekend worden getekend in de SPACE FILLING tekenstijl. Het tekenen van bindingen wordt gedaan in de *drawBonds* methode van de *Crysdata* klasse. Deze methode zal in eerste instantie een béziercirkel tekenen en het de naam *bez* toekennen, deze cirkel zal gebruikt worden als vorm waarmee de *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.

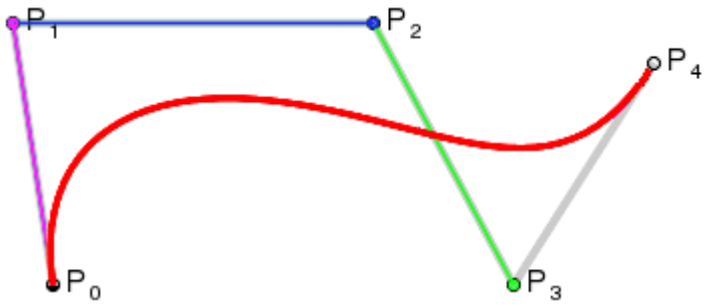
Enkel wanneer deze afstand kleiner is dan diegene die de gebruiker kiest moeten we een binding tekenen tussen de atomen. Hiervoor wordt de *makeBond* methode opgeroepen met de twee te binden atomen als argument. Deze methode gaat eerst met de *bpy.data* module de twee bollen zoeken die de atomen voorstellen. Deze twee object 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 onze 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 bézierkromme maken. Een bézierkromme is een manier waarop lijnen wiskundig kunnen worden voorgesteld aan de hand van twee of meer punten, die de graad van de bézierkromme 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 bézierkromme 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 bézierkromme met graad 3 (Tregoning, 2007)

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

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

We kennen onze bézierpunten toe aan variabelen zodat deze later makkelijk kunnen worden opgeroepen.

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

Nu gaan we van onze kromme een object maken in de Blender omgeving. Aan zo een object kunnen we vervolgens *modifiers* toevoegen aan dit object. In ons geval gaan we gebruikmaken 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 van aan onze atomen, maar nog niet aan onze kromme.

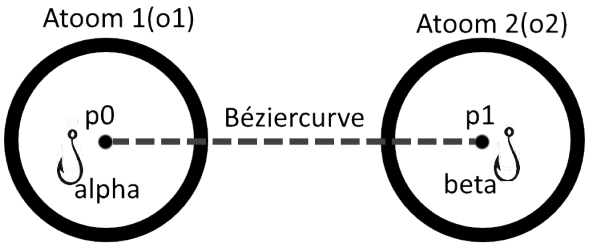
```
bpy.context.collection.objects.link(obj)
bpy.context.view_layer.objects.active = obj

bpy.ops.object.mode_set(mode='EDIT')
```

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 bézierpunten van onze kromme apart kunnen selecteren.

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

Herinner dat we eerder onze twee bézierpunten 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.



Modifier	Object	Bézierpunt
alpha	o1	p0
beta	o2	p1

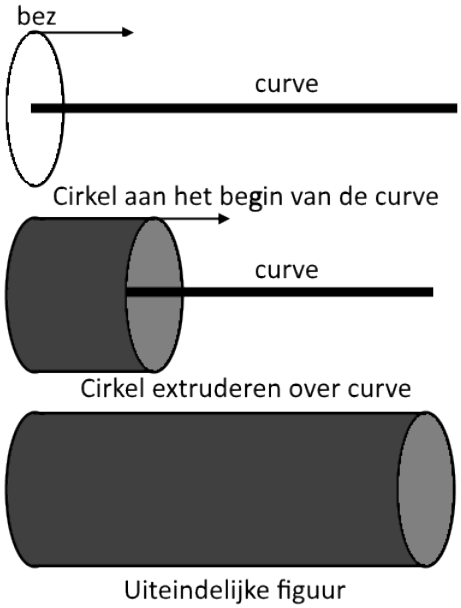
Figuur 4.2: Overzicht van de *hook modifier*

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*.

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

In het begin van deze sectie hebben we een béziercirkel, *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 betekend: 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

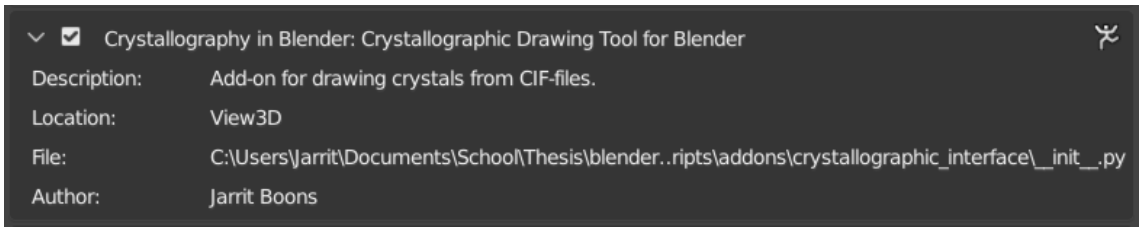
4.4.1 Add-on header

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* is onder het *add-on* tabblad. We merken op dat beide het *name* 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 = {                                                          103
    "name": "Crystallographic Drawing Tool for Blender",             104
    "description": "Add-on for drawing crystals from CIF-files.",    105
    "author": "Jarrit Boons",                                        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 heb is er voor *Crystallographic Drawing Tool for Blender*, afgekort naar *CDTB*, gekozen. De catergorie 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

Zoal 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.

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

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 wanneer de operator wordt uigevoerd er een filebrowser opent, met de *return* op lijn 132 zorgt Blender 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 de leesbaar is vanuit heel het programma, en stopt de operator.

Bij het maken van de tweede operatorklasse komt iets meer 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].

```
210 bpy.types.Scene.bond_distance = bpy.props.FloatProperty(
211     name="Bond distance",
212     description="Set max distance for bonds to occur",
213     default=2,
214     min=0.0,
215     max=10.0,
216     precision=2
```

```
)                                                                    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* kan 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 toekenen 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 gebruiken 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 zal 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. Listing[4.16] toont een deel van de *draw* methode waarin we het paneel creëren door een reeks van deze methodes in een bepaalde volgorde op te roepen.

```
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('\\', 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

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.

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 pystarrw. `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.

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

Bijlagen (op USB)

- Bijlage A: De CIF-dictionary
- Bijlage B: Cif-bestand van de kristalstructuur van Chaziet
- Bijlage C: Ontworpen programma dat CIF-bestanden kan inlezen
- Bijlage D: Volledige code van het programma

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

Grantt Map: Actuele Planning

	Sep	Okt	Nov	Dec	Jan	Feb	Maa	Apr	Mei	Jun
Literatuurstudie										
Kristalografie										
Blender										
Libraries										
Uitvoering										
Programmastructuur										
Schrijven scripts										
Debugging										
Optimaliseren										
Extra features (Add-ons)										
Voorbereiding Thesis										
Schriftelijk										
Mondeling										

CHECKLIST bij masterproef (bij tussentijds verslag)

Volgende checklist is een hulpmiddel om na te gaan of alle elementen in het tussentijds verslag opgenomen zijn. De ingevulde en ondertekende lijst wordt bij de promotor afgegeven samen met de twee kopijen van het tussentijds verslag.

- ☑ titel, naam van de student en promotor
- ☑ inhoudsopgave
- ☑ lijst met afkortingen (alfabetisch) en symbolen
- ☑ situering
- ☑ probleemstelling
- ☑ doelstelling
- ☑ onderzoeksvraag (en eventuele deelvragen)
- ☑ literatuurstudie van de betrokken state of the art
- ☑ reeds gerealiseerde elementen
- ☑ geactualiseerde planning (Gantt kaart)
- ☑ bibliografie (met correcte verwijzingen in de tekst)

De verantwoordelijkheid voor het correct invullen van deze checklist ligt bij de student.

Naam	datum	handtekening
Jarrit Boons	20/12/2018