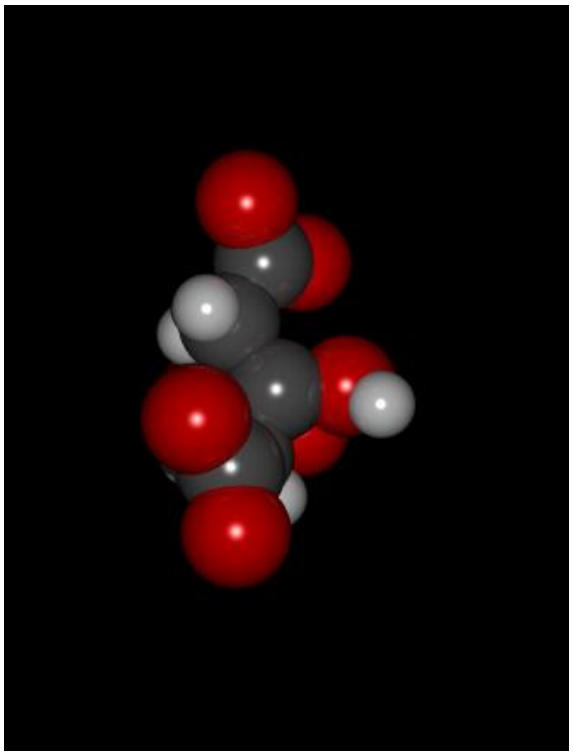
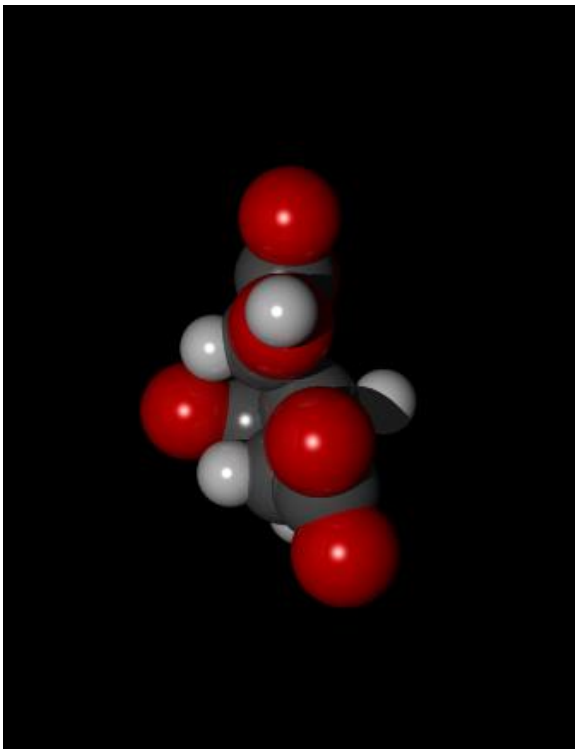
# Citroenzuurcyclus

Citraat naar Isocitraat





Lisa Hu

Maartje van der Hulst

Loes Oldhoff – OLLO

Annemarie Akker – AKAN

BFV2

24-01-202

# Citroenzuurcyclus

Citraat naar Isocitraat

Hanzehogeschool Groningen

Bio-informatica (ILST)

Lisa Hu – 414264

Maartje van der Hulst – 374361

Loes Oldhoff – OLLO

Annemarie Akker – AKAN

24-01-2022

# Samenvatting

Dit project is vooral gericht op het inzichtelijk maken van een biologisch proces of chemische reactie wegens een simulatie. Hiervoor is verdiept in de tweede stap van de citroenzuurcyclus: Citraat naar isocitraat. Citraat speelt in het lichaam een grote rol voor meerdere processen. Het project is samengesteld met behulp van een Python library genaamd pypovray. (Kempenaar, 2019) Het visualiseren van dit proces is gelukt, maar het script runnen via SLURM gaf bepaalde problemen bij het renderen van het laatste deel van de simulatie. Hier liggen dus ook de verbeterpunten.

# Inhoudsopgave

# Figurenlijst

# Inleiding

De citroenzuurcyclus speelt een belangrijke rol in het lichaam en is een deel van het proces dat glucose omzet naar energie, water en koolstofdioxide. De cyclus vindt plaats in de matrix van de mitochondriën. De cyclus bestaat uit 10 stappen, maar tijdens de tweede stap is waar citraat omgezet wordt naar isocitraat met behulp van het enzym aconitase. (Wat is de citroenzuurcyclus?, 2020)

Citraat is belangrijk voor het lichaam om gezondheidsproblemen te voorkomen. Een voorbeeld hiervan is dat citraatkristallen de vorming van nierstenen kunnen voorkomen. (Endocrinologie, 2017) Ook kan citraat worden gebruikt als bloedverdunner, wat vooral goed effect heeft op patiënten met ernstig nierfalen. Dit werkt beter dan andere bloedverdunners, omdat het de kans op bloedingen ergens anders in het lichaam niet vergroot zoals andere bloedverdunners dat doen. (Bloedverdunner citraat effectiever en veiliger bij ernstig zieke nierpatiënten, 18)
Het enzym aconitase wat citraat omzet naar isocitraat kan defect zijn. Dit wordt vaak in verband gebracht met mitochondriale ziekten. (Portnov, sd) Verder kan bij een afwijking in het ijzer-zwavelcluster in het enzym aconitase ervoor zorgen dat dit enzym niet meer goed werkt. Dit heeft als gevolg dat er veel citraat ophoopt in de cel. Die ophoping wordt door de cel omgezet in vet. Een ophoping van vet in cellen kan voor allerlei problemen zorgen zoals niet-alcohol gerelateerde leververvetting. (Onderzoek naar ijzer-zwavelcluster biedt nieuwe mogelijkheden om ziekten te onderzoeken, sd)

Naast de citroenzuurcyclus is citraat ook belangrijk voor de vetzuursynthese.

Het visualiseren kan ervoor zorgen om inzicht te creëren in de reactie, door het mechanisme van de reactie te laten zien. Dit kan focus geven voor eventueel verdere onderzoeken.

In de volgende hoofdstukken wordt beschreven wat er is gebruikt voor het maken van de simulatie en hoe deze tot stand is gekomen. Daarna is door afbeeldingen van de simulatie de

Resultaten weergegeven. Daarop volgen de Discussie en de Bijlage A: Code met de code en de default.ini.

# Materiaal en Methode

## Materiaal

Dit project is opgezet op basis van programmeren, specifiek in de taal *Python3*. Deze programmeertaal kan vooral gebruikt worden voor het bouwen van software op verschillende domeinen (bijvoorbeeld web applicaties of in de data wetenschappen). Python3 is in verschillende versies verkrijgbaar. Python 3.9.2 of nieuwer wordt aangeraden, omdat de productie achter de oudere versies mogelijk zijn stopgezet. Deze taal is uit te breiden met vele packages en libraries. De gebruikte Python packages zijn als volgt:

| **Package** | **Versie** | **Beschrijving** |
|---|---|---|
| Python (Rossum, 2022) | 3.9.2 (en nieuwer) | De basis van het project |
| pypovray (Kempenaar, 2019) | - | Creëren van animaties met POVRAY als leidende code (Hallam Oaks, 2003) |
| vapory (Zulko, 2018) | 0.1.2 | Library om fotorealistische 3D-scènes te renderen met behulp van POV-Ray |
| MoviePy (moviepy, 2022) | 1.0.1 | Package voor het bewerken van een filmpje |
| NumPy (Oliphant, 2022) | 1.17.4 | Package voor wiskundige modules |
| pathos (pathos, 2022) | 0.2.5 | Framework package om taken parallel te laten lopen |

Voor de code van dit project, zie Bijlage A: Code.

Naast de Python packages, gebruikt het project ook een programma genaamd *SLURM*. Dit programma is een *open source scheduler* voor Linux servers. Dit programma zorgt ervoor dat verschillende taken van een programma worden toegewezen naar verschillende werkstations, beschikbaar op de server. Het is vergelijkbaar met parallel programmeren. (Sales, 2021)
SLURM werkt via een *bash script* (zie Bijlage C: SLURMshellscript.sh) en is uitvoerbaar via de command line.

Voor het creëren van de moleculen, wordt er gebruik gemaakt van PDB-bestanden. Citraat en isocitraat zijn verkregen van het 2J80 eiwit en 1CW7 eiwit, respectievelijk. (Usher, 1997-12-24 ) (Stroud, 1999) Met behulp van *ArgusLab* (Thompson, 2000) zijn deze PDB-bestanden bewerkt om de waterstof atomen die ontbraken toe te voegen aan het bestand.

## Methode

De citroenzuurcyclus is een proces van 10 stappen lang en vindt zich in de mitochondriën plaats. In de tweede stap wordt citraat gebruikt om de cyclus te initiëren om als uiteindelijke product weer citraat in meervoud te vormen. Deze tweede stap zorgt ervoor dat citraat eerst wordt omgezet tot isocitraat. De chemische reactie luidt als volgt: (KEGG, Reaction: R01324, 2022)

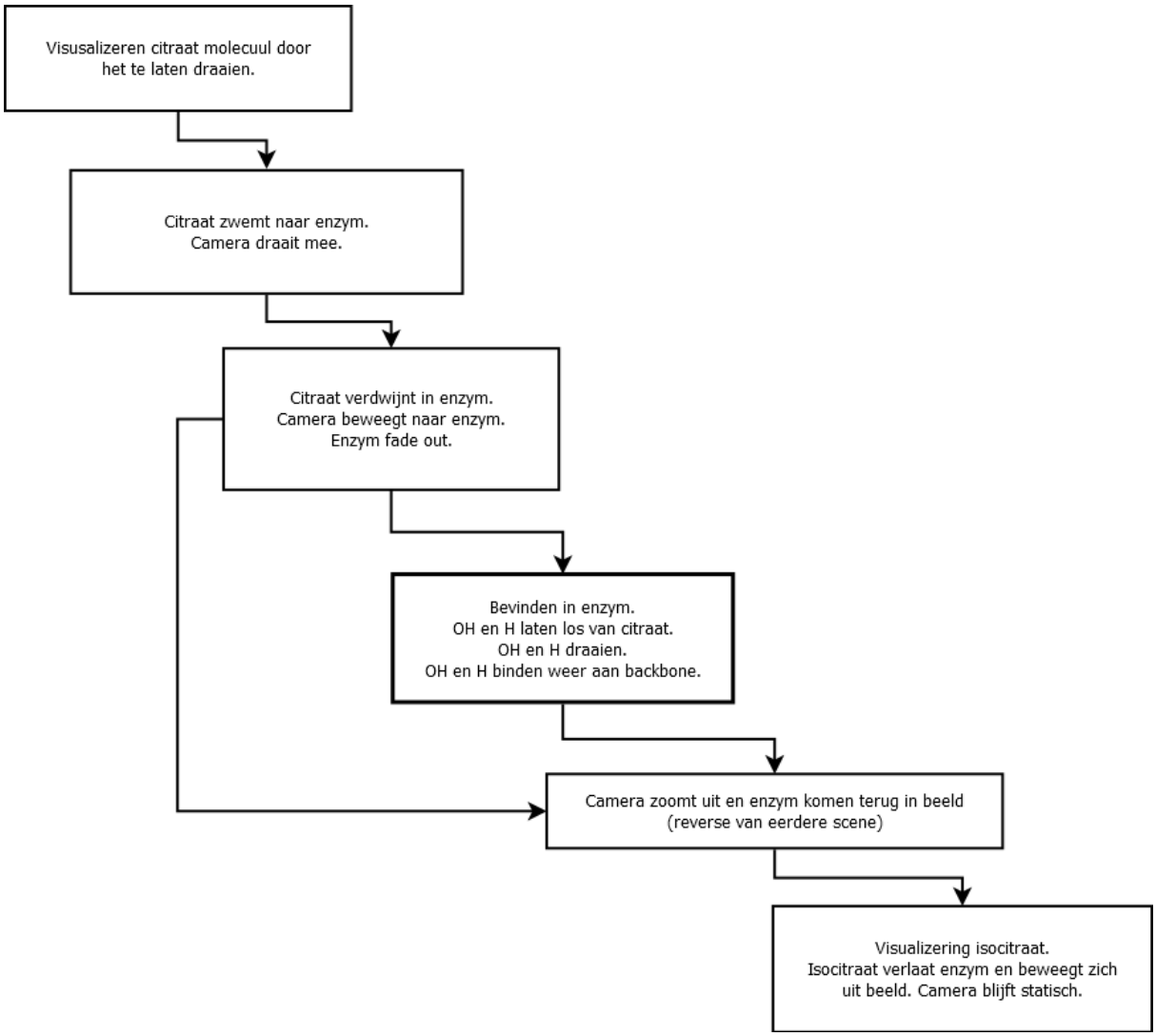
Citraat → cis-Aconitaat + $H_2O$ (aconitase) → Isocitraat (aconitase)

Aconitase is een hydrolyase (KEGG, EC 4.2.1.3, 2022) die ervoor zorgt dat er een OH-groep en een H-atoom worden afgesplitst en met elkaar worden gewisseld voor binding. Hiermee wordt isocitraat gevormd: een isomeer van citraat.

Het gebruik van de pypovray library wordt in de library uitgebreid uitgelegd.

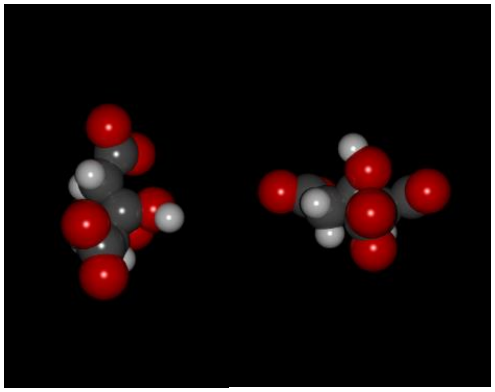
## Resultaten

Om het proces van het project makkelijker in te zien, is er een flowchart gemaakt over het verloop van de animatie. De grote lijn in de animatie zou het visualiseren van de verschillende moleculen zijn en het mechanisme achter de reactie van citraat naar isocitraat.

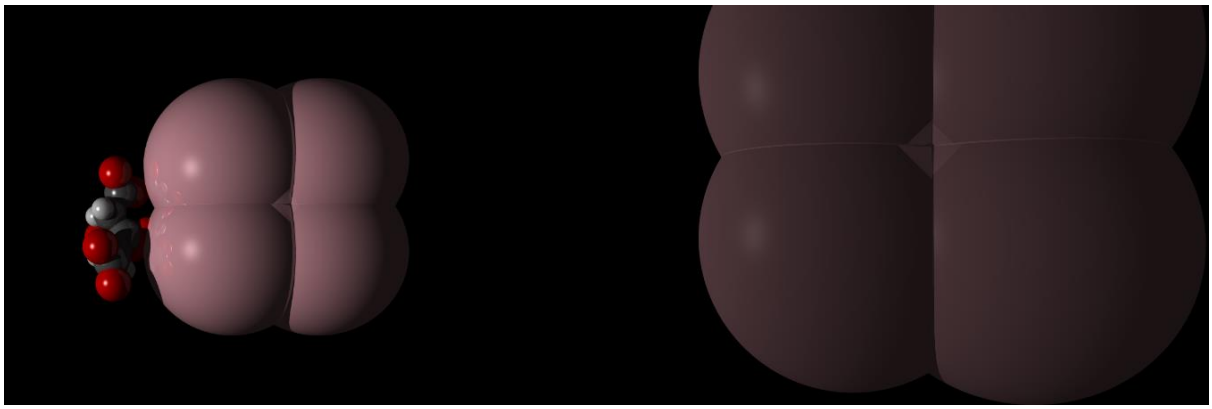



*Figuur 1: Flowchart*

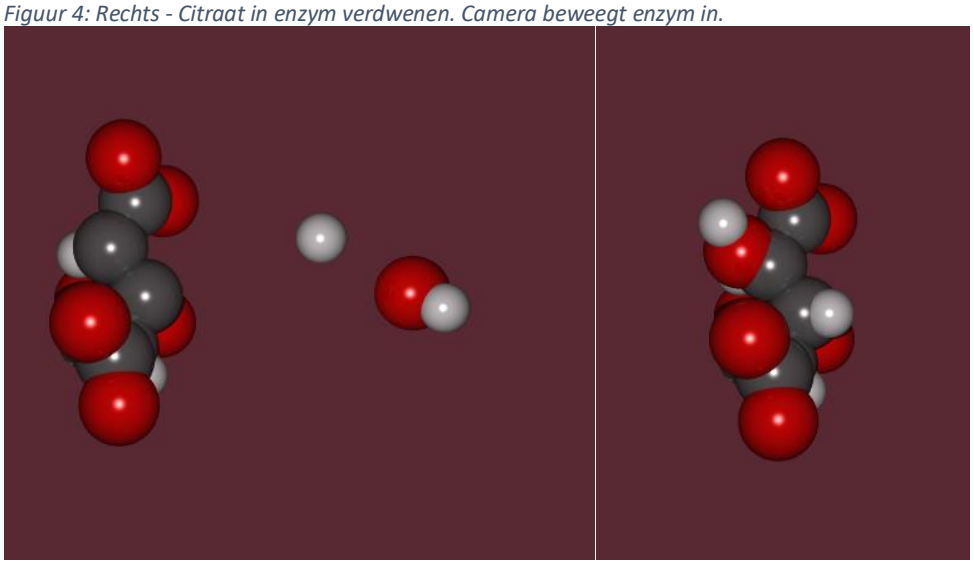
De code is op dezelfde volgorde geschreven als de flowchart. De pijl van de derde scène naar de vijfde scène duidt erop dat de code en onderbouwing gelijk zijn, maar elkaars achterstevoren beeld zijn. Zie Bijlage A: Code, regels 119-315 (class PovRayScenes). Hieronder een paar beelden van het eindresultaat:



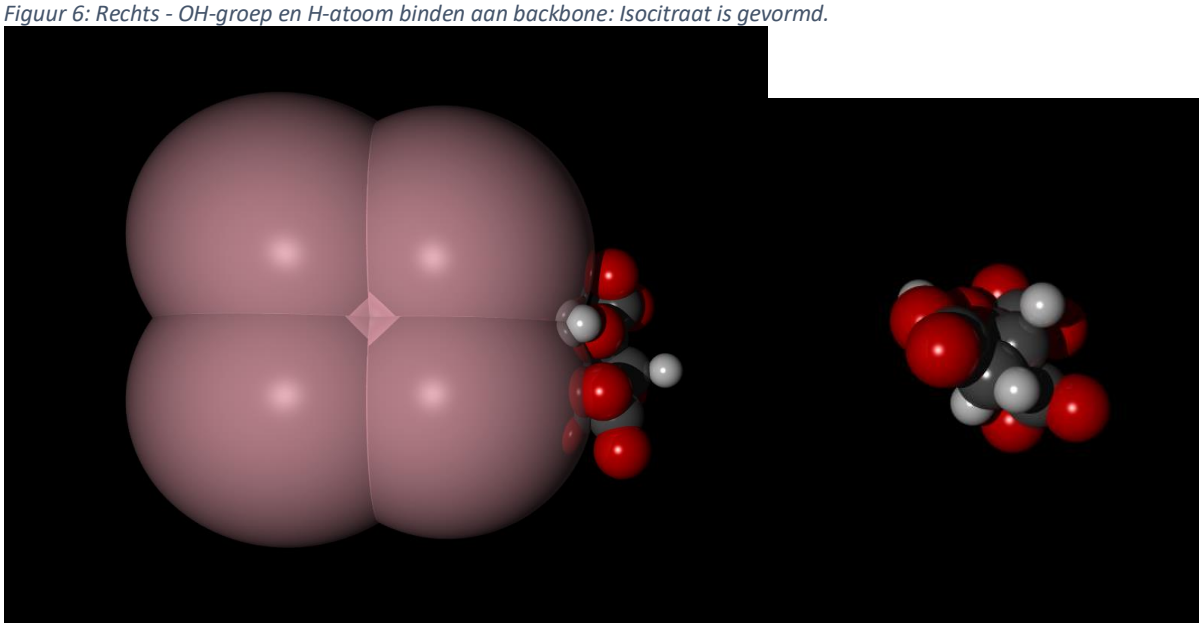
*Figuur 2: Draaiing van citraat. Links het eerste frame. Rechts het molecuul tijdens het draaien.*



*Figuur 3: Links - Citraat verdwijt enzym in.*
*Figuur 4: Rechts - Citraat in enzym verdwenen. Camera beweegt enzym in.*



*Figuur 5: Links - OH-groep en H-atoom splitsen af.*
*Figuur 6: Rechts - OH-groep en H-atoom binden aan backbone: Isocitraat is gevormd.*



*Figuur 7: Links - Isocitraat beweegt molecuul uit.*
*Figuur 8: Isocitraat mid draai ter visualisatie.*

## Discussie

Het gebruik van SLURM heeft het verwerken van het project aanzienlijk versneld. Helaas werkt het programma niet wanneer men de 1000 taken overschreid. Hier liepen wij tegen aan en hierdoor is het uiteindelijke product in volgorde misgelopen: Frames 1000 tot 1050 worden vermengd met de frames tussen de 100 en 105 waardoor de simulatie een vreemde skip heeft rond de 4$^{e}$ seconde.

- Mogelijke oplossing hiervoor zou zijn om het niet via SLURM te runnen. De pypovray library biedt een mogelijkheid om de code tot mp4 te renderen, waardoor de frames wel op de juiste volgorde gerenderd zouden moeten worden. Hiervoor zou het gedeelte van de boilerplate code veranderd moeten worden (zie Bijlage A: Code, vanaf regel 351) naar de volgende line:
  ```
  pypovray.render_scene_to_mp4(main)
  ```
  De code is zo geschreven dat de UsePool optie van toepassing is voor een snellere render (zie Bijlage B: default.ini, regel 24)

# Bibliografie


*Bloedverdunner citraat effectiever en veiliger bij ernstig zieke nierpatiënten*. (18, 7 2016). (ziekten.nl) Opgeroepen op 11 2021, 26, van https://www.ziekten.nl/bloedverdunner-citraat-effectiever-en-veiliger-bij-ernstig-zieke-nierpatienten/

Endocrinologie. (2017, 9). *Nierstenen*. (Leids Universitair Medisch Centrum) Opgeroepen op 11 2021, 26, van https://www.lumc.nl/patientenzorg/praktisch/patientenfolders/nier-stenen

*ffmpy*. (2022, 1 24). Opgehaald van pypi: https://pypi.org/project/ffmpy/

Hallam Oaks, P. L. (2003). *POV-Ray*. Opgehaald van http://www.povray.org

KEGG. (2022). *EC 4.2.1.3.* Kyoto, Japan: KEGG: Kyoto Encyclopedia of Genes and Genomes. Opgehaald van https://www.kegg.jp/entry/4.2.1.3

KEGG. (2022). *Reaction: R01324.* Kyoto, Japan: KEGG: Kyoto Encyclopedia for Genes and Genomes. Opgehaald van https://www.kegg.jp/entry/R01324

Kempenaar, M. (2019, 11 14). *BitBucket*. Opgeroepen op 11 25, 2021, van https://bitbucket.org/mkempenaar/pypovray/src/master/

*moviepy*. (2022, 1 24). Opgehaald van PyPi: https://pypi.org/project/moviepy/

Oliphant, T. (2022, 1 24). *welcome*. Opgehaald van Numpy: https://numpy.org/

*Onderzoek naar ijzer-zwavelcluster biedt nieuwe mogelijkheden om ziekten te onderzoeken*. (sd). (TH wetenschap) Opgeroepen op 11 2021, 26, van http://nl.scienceaq.com/Chemie/1007082123.html

*pathos*. (2022, 1 24). Opgehaald van PyPi: https://pypi.org/project/pathos/

Portnov, A. (sd). *Mitochondiale ziekten als gevolg van aandoeningen van de Krebs-cyclus*. (i live ok) Opgeroepen op 11 26, 2021, van https://nl.iliveok.com/health/mitochondriale-ziekten-als-gevolg-van-aandoeningen-van-de-krebs-cyclus_76438i15937.html

Rossum, G. v. (2022, 1 24). *Welcome*. Opgehaald van Python: https://www.python.org/about/help/

Sales, S. (2021, August 6). *SLURM*. Opgeroepen op January 10, 2022, van https://slurm.schedmd.com/overview.html

Stroud, M. F.-M. (1999, 9 1). *1CW7*. (Protein data bank) Opgeroepen op 1 12, 2022, van https://files.rcsb.org/view/1CW7.pdb

Thompson, M. (2000, October 14). *ArgusLabs*. Opgeroepen op December 17, 2021, van http://www.arguslab.com/arguslab.com/ArgusLab.html

Usher, K. R. (1997-12-24 , 12 24). *6CSC*. (Protein data bank) Opgeroepen op 1 2022, 12, van https://files.rcsb.org/view/6CSC.pdb

*Wat is de citroenzuurcyclus?* (2020, 1 8). (Greelane) Opgeroepen op 1 12, 2022, van https://www.greelane.com/nl/science-tech-math/wetenschap/citric-acid-cycle-p2-603894/

Zulko. (2018, September 17). *GitHub*. Opgeroepen op November 25, 2021, van https://github.com/Zulko/vapory

# Bijlage A: Code

```
#!/usr/bin/python3

"""
This program renders an animation with the use of the PyPovRay library.
The animation with visualize the mechanism of the second step in the citric acid cycle:
    citrate to isocitrate in the enzyme aconitase.
"""

__author__ = "Lisa Hu & Maartje van der Hulst"
__date__ = 2021.12
__version__ = 1.0

# IMPORTS
import sys
from math import pi
from pypovray import pypovray, pdb, SETTINGS, models
from vapory import Scene, Texture, Pigment, Finish, Sphere, LightSource, Camera, Background
import datetime


# OBJECTS
class PovRayObjects:
    """
    Module to make the objects used in PovRay movie
    """
    def __init__(self):
        self.citrate, self.isocitrate = self.get_molecules()
        self.enzyme = self.make_enzyme()

    @staticmethod
    def make_enzyme():
        """
        Enzyme object
        """
        # Style model for the spheres
        sphere_style = Texture(Pigment('color', [1, 0.7, 0.75], 'filter', 0),
                               Finish('phong', 0.2, 'reflection', 0.3))
        # Spheres
        sphere1 = Sphere([30, -1, 0], 5, sphere_style)  # Bottom left
        sphere2 = Sphere([35, -1, 0], 5, sphere_style)  # Bottom right
        sphere3 = Sphere([30, 4, 0], 5, sphere_style)  # Top left
        sphere4 = Sphere([35, 4, 0], 5, sphere_style)  # Top right

        return [sphere1, sphere2, sphere3, sphere4]

    @staticmethod
    def get_molecules():
        """
        Chemical components
        """
        citrate = pdb.PDBMolecule(f"{SETTINGS.AppLocation}pdb/citrate_new.pdb",
                                  center=True, offset=[0, 0, 0])
        isocitrate = pdb.PDBMolecule(f"{SETTINGS.AppLocation}pdb/isocitrate_new.pdb",
                                     center=True, offset=[0, 0, 0])




# FUNCTIONS
class PovRayFunctions:
    """
    Module for functions used in PovRay movie
    """
    def __init__(self, tf_end):
        """
        Initializing function
        :param tf_end: List of last frames per scene
        """
        self.tf_end = tf_end

    def get_timesframes(self):
        """
        Create the lists of start frames and duration frames
        """
```

```
        tf_start = []
        start_point = 0
        for i in range(len(self.tf_end)):
            # i = [0, 1, 2...]
            tf_start.append(start_point)
            # start_point = [120, 240, 420...]
            start_point = self.tf_end[i]
            # tf_start = [0, 120, 240, 420...]

        # i = index of the start list
        # Subtracting the end time with start time results in duration time
        tf_dur = [self.tf_end[i] - tf_start[i] for i in range(len(tf_start))]

        return tf_start, tf_dur

    @staticmethod
    def get_distance(step, duration, distance, start_time=0):
        """
        Get distances per frame
        :param step: Step in the scene
        :param duration: Duration of the scene in seconds
        :param distance: [x, y, z] vector
        :param start_time: Start of the movement in the scene, default 0
        """
        total_frames = SETTINGS.RenderFPS * duration
        first_frame = (step + 1) - SETTINGS.RenderFPS * start_time
        distances = [x / total_frames * first_frame for x in distance]

        return distances

    @staticmethod
    def rotate_molecule(rotation, molecule, duration, step):
        """
        Function to let the molecule rotate
        :param rotation: Amount of rotation in radials
        :param molecule: Molecule to rotate
        :param duration: Duration of the rotation in frames
        :param step: Step in the rotation
        """
        rads = (rotation * pi / duration) * step
        molecule.rotate([1, 1, 0], rads)


# SCENES
class PovRayScenes:
    """
    Scenes used for the povray movie
    """

    def __init__(self, pr_objs, pr_funcs, tf_start, tf_dur, tf_end):
        """
        Initializing function
        :param pr_objs: PovRayObjects class
        :param pr_funcs: PovRayFunctions class
        :param tf_start: List of frames when the scenes start
        :param tf_dur: List of amounts of frames per scene
        :param tf_end: List of last frame per scene
        """
        self.probj = pr_objs
        self.prfunc = pr_funcs
        self.start = tf_start
        self.dur = tf_dur
        self.end = tf_end

    def s1_citrate_rotation(self, step):
        """
        First scene: Rotating citrate for visualization
        """
        # Create camera and light source
        camera = Camera('location', [0, 0, -30], 'look_at', [0, 0, 0])
        lighting = LightSource([0, 0, -20], 'color', [1, 1, 1])
        # Get the citrate molecule
        citrate = self.probj.citrate
        # Let citrate rotate
        self.prfunc.rotate_molecule(2, citrate, self.dur[0] * 30, step)
        # List of objects to render
        objects = citrate.povray_molecule + [lighting]
```

```
        return Scene(camera, objects=objects)

    def s2_moving(self, step):
        """
        Second scene: Moving citrate into the enzyme
        """
        # Create light source
        lighting = LightSource([0, 0, -20], 'color', [1, 1, 1])

        # Get the position of the camera
        position = self.prfunc.get_distance(step, self.dur[1], [31, 0, 0])
        # Set the z position backwards
        position[2] = -30
        # Get the position of citrate and the camera look position
        looking = self.prfunc.get_distance(step, self.dur[1], [31, 0, 0])
        camera = Camera('location', position, 'look_at', looking)

        # Get citrate and enzyme
        enzyme = self.probj.enzyme
        citrate = self.probj.citrate
        citrate.move_offset(looking)

        # List of objects to render
        objects = citrate.povray_molecule + enzyme + [lighting]

        return Scene(camera, objects=objects)

    def s3_fading_in(self, step):
        """
        Third scene: Camera moves 'into' the enzyme, light fades
        :return:
        """
        # Get the camera location
        position = self.prfunc.get_distance(step, self.dur[2], [0, 0, 25])
        position[0] = 32
        position[2] -= 30
        camera = Camera('location', position, 'look_at', [30, 0, 0])
        # Let the light fade out
        intensity = self.prfunc.get_distance(step, self.dur[2], [-1, -1, -1])
        # Outcome of function above is negative -> set positive
        intensity[:] = [number + 1 for number in intensity]
        # Light source that fades out
        lighting = LightSource([0, 0, -20], 'color', intensity)

        # Get enzyme object
        enzyme = self.probj.enzyme
        # List of objects to render
        objects = enzyme + [lighting]

        return Scene(camera, objects=objects)

    def s4_switching(self, step):
        """
        Fourth scene: Located inside the enzyme, visualizing the mechanism of the reaction
        :return:
        """
        # Get the objects
        background = Background('color', [0.35, 0.16, 0.2])
        citrate = self.probj.citrate
        camera = Camera('location', [0, 0, -30], 'look_at', [0, 0, 0])
        lighting = LightSource([0, 0, -20], 'color', [1, 1, 1])

        # Split respective OH and H atoms from citrate and move away
        if step <= 90:
            offset = self.prfunc.get_distance(step, self.dur[3] / 3, [3, 0, 0])
            oh_group = citrate.divide([5, 15], 'oh_group', offset=offset)  # Split OH
            h_atom = citrate.divide([13], 'h_atom', offset=offset)  # Split H

        # Switch OH and H atoms + rotate OH group
        elif step <= 180:
            oh_group = citrate.divide([5, 15], 'oh_group', offset=[6, 0, 0])
            h_atom = citrate.divide([13], 'h_atom', offset=[6, 0, 0])
            # Distances for moving the OH and H up and down
            moving_up = self.prfunc.get_distance(step, self.dur[3] / 3, [0, 2, -0.55], 3)
            moving_down = self.prfunc.get_distance(step, self.dur[3] / 3, [0, -1.5, 0.39], 3)
            # Move H atom
```

```
            h_atom.move_offset(moving_down)
            # Rotate OH group while moving
            oh_group.move_offset(moving_up)
            self.prfunc.rotate_molecule(0.5, oh_group, (self.dur[3] / 3) * 30, step - 90)

        # Move OH and H atoms back towards molecule
        else:
            oh_group = citrate.divide([5, 15], 'oh_group', offset=[6, 2.1, -0.6])
            h_atom = citrate.divide([13], 'h_atom', offset=[6, -1.5, 0.39])
            self.prfunc.rotate_molecule(0.5, oh_group, (self.dur[3] / 3) * 30, 90)  # Rotation
OH
            # Distance for moving OH and H inwards
            inwards_oh = self.prfunc.get_distance(step, self.dur[3] / 3, [-9.5, 0, 0], 6)
            inwards_h = self.prfunc.get_distance(step, self.dur[3] / 3, [-4, 0, 0], 6)
            # Move OH and H inwards
            oh_group.move_offset(inwards_oh)
            h_atom.move_offset(inwards_h)

        # List of objects to render
        objects = citrate.povray_molecule + oh_group.povray_molecule + h_atom.povray_molecule
+ \
                  [background] + [lighting]

        return Scene(camera, objects=objects)

    def s5_fading_out(self, step):
        """
        Fifth scene: Isocitrate is made, light fades out, moving out of the enzyme
        """
        # Dim lights inside the enzyme
        if step <= 60:
            # Get the objects
            isocitrate = self.probj.isocitrate
            background = Background('color', [0.35, 0.16, 0.2])
            camera = Camera('location', [0, 0, -30], 'look_at', [0, 0, 0])
            # Light fading
            intensity = self.prfunc.get_distance(step, self.dur[4] / 3, [-1, -1, -1])
            intensity[:] = [number + 1 for number in intensity]  # Same negative error
            # Light fade out
            lighting = LightSource([0, 0, -20], 'color', intensity)
            # List of objects to render
            objects = [background] + [lighting] + isocitrate.povray_molecule

        # Move out of the enzyme
        else:
            # Get the objects
            enzyme = self.probj.enzyme
            lighting = LightSource([30, 0, -20], 'color', [1, 1, 1])
            # Move the camera backwards, out of the enzyme
            position = self.prfunc.get_distance(step, self.dur[4], [0, 0, -30], 2)
            position[0] = 35
            position[2] -= 5
            camera = Camera('location', position, 'look_at', [30, 0, 0])
            # List of objects to render
            objects = enzyme + [lighting]

        return Scene(camera, objects=objects)

    def s6_final(self, step):
        """
        Sixth and final scene: Move isocitrate out of the enzyme,
        rotation of isocitrate for visualization
        """
        # Get the objects
        isocitrate = self.probj.isocitrate
        lighting = LightSource([30, 0, -20], 'color', [1, 1, 1])
        enzyme = self.probj.enzyme

        # Move isocitrate out of the enzyme, camera focuses on isocitrate
        if step <= 60:
            position = self.prfunc.get_distance(step, self.dur[5] / 3, [20, 0, 0])
            position[0] += 31  # Set start position of isocitrate at [31, 0, 0]
            isocitrate.move_offset(position)
            camera = Camera('location', [35, 0.0, -25], 'look_at', position)

        # Rotation isocitrate for visualization
        else:
```

```
            isocitrate.move_offset([51, 0, 0])
            camera = Camera('location', [35, 0.0, -25], 'look_at', [51, 0, 0])
            self.prfunc.rotate_molecule(2, isocitrate, ((self.dur[5] / 3) * 2) * 30, step -
60)


        # List of objects to render
        objects = isocitrate.povray_molecule + enzyme + [lighting]

        return Scene(camera, objects=objects)


def main(step):
    """
    Main function
    """
    tf_end = [120, 240, 420, 690, 870, 1050]
    # Scenes   1    2    3    4    5     6

    # Initialize functions
    probj = PovRayObjects()
    prfunc = PovRayFunctions(tf_end)
    # Get the time frames
    tf_start, tf_dur = prfunc.get_timesframes()
    tf_dur[:] = [number / 30 for number in tf_dur]
    # Initialize the scenes
    prscenes = PovRayScenes(probj, prfunc, tf_start, tf_dur, tf_end)

    # Create the scenes according to step
    if step <= tf_end[0]:
        scene = prscenes.s1_citrate_rotation(step)
    elif step <= tf_end[1]:
        scene = prscenes.s2_moving(step - tf_end[0])
    elif step <= tf_end[2]:
        scene = prscenes.s3_fading_in(step - tf_end[1])
    elif step <= tf_end[3]:
        scene = prscenes.s4_switching(step - tf_end[2])
    elif step <= tf_end[4]:
        scene = prscenes.s5_fading_out(step - tf_end[3])
    else:
        scene = prscenes.s6_final(step - tf_end[4])
    return scene


if __name__ == "__main__":
    for i in range(1000, 1050):
        pypovray.render_scene_to_png(main, i)

    pypovray.render_scene_to_png(main, int(sys.argv[1]))
```

# Bijlage B: default.ini

```
; See the 'Configuring' section in the following document for a per-setting
explanation:
;
http://nbviewer.jupyter.org/urls/bitbucket.org/mkempenaar/pypovray/raw/master/manua
l/install_and_configure.ipynb

[GENERAL]
; General application settings.
AppLocation = /homes/ljbhu/thema2/project/pypovray/
OutputPrefix = simulation
; Remove the "%(AppLocation)s" from the paths below to change to relative paths
OutputImageDir = %(AppLocation)s/images
OutputMovieDir = %(AppLocation)s/movies
; Log-level: DEBUG, INFO (default), WARNING, ERROR and CRITICAL
LogLevel = INFO

[RENDER]
; Rendering settings influencing the output format and quality
ImageWidth = 1600
ImageHeight = 900
Quality = 9
AntiAlias = 0.01
UsePool = True
Workers = 8

[SCENE]
; Scene settings controlling the duration and frames per second
; for the animation. The RenderFPS is used in conjunction with the
; duration to get the total amount of frames to render. The MovieFPS
; is only used for the ffmpeg encoding.
Duration = 60
RenderFPS = 30
FrameTime = 1 / %(RenderFPS)s
NumberFrames = %(Duration)s * %(RenderFPS)s
MovieFPS = 30

[OTHER]
; Show each rendered frame in a popup
ShowWindow = False
; Remove all temporary generated data after rendering
RemoveTempFiles = False
```

# Bijlage C: SLURMshellscript.sh

```
#!/bin/bash

# Example of running python script with a job array

#SBATCH -J Lisa_final                # Job name
#SBATCH -p workstations              # Partition. workstations or assemblix. See sinfo
#SBATCH --array=1-999                # how many tasks in the array
#SBATCH -c 1                         # one CPU core per task
#SBATCH -t 7:00                      # Time limit per job
#SBATCH --chdir /homes/ljbhu/thema2/project/pypovray                # Working dir
#SBATCH -o /homes/ljbhu/thema2/project/pypovray/slurms/output.%a.out          # STDOUT

# Run python script with a command line argument
# srun deals with using the slurm reservation, making the relevant hardware available to the
script/program to run, and if applicable, MPI communication
srun python3 eindopdracht_LisaHu_MaartjevdHulst.py $SLURM_ARRAY_TASK_ID

#Run in terminal met SBATCH
```