Het Kunstgalerijprobleem in $\mathbb{R}^n$

Dennis Geelhoed, s4785517

Begeleider: Dr. W. Bosma
Tweede lezer: Dr. H. Don

B.Sc. Wiskunde

Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Radboud Universiteit Nijmegen

31 Augustus 2020

1 Introductie

Tegenwoordig is het vakgebied Computationele Meetkunde overal te vinden. We leven in een tijd waarin computers en simulaties een grote rol spelen. Voor ontelbaar veel computerprogramma's zijn berekeningen aan lijnen, driehoeken en vlakken nodig. Een klassiek probleem in de computationele meetkunde is het kunstgalerijprobleem uit 1973. De vraag: "Hoeveel bewakers heb je nodig om een kunstgalerij te bewaken?"klinkt misschien specifiek, maar de methode om dit probleem op te lossen is breed toepasbaar.

In dit onderzoek bekijken we dit probleem vanuit een andere hoek dan gebruikelijk. De motivatie hiervoor is dat het kunstgalerijprobleem zich drastisch anders gedraagt in 3 dimensies of hoger. Dit maakt dat vele van de bekende stellingen over kunstgalerijen niet te generaliseren zijn naar 3 dimensies, laat staan naar n dimensies. Uit de wiskunde is het bekend dat een generalisatie van een probleem diepere inzichten kan bieden.

Deze benadering van dit onderzoek is meer analytisch van aard. De lijnstukken waaruit een 2D kunstgalerij is opgebouwd worden gegeneraliseerd tot simplices. De kunstgalerij, die de vorm heeft van een veelhoek, zal worden gegeneraliseerd tot polytoop. Vervolgens zullen we allerlei algoritmes beschrijven die snijpunten en projecties van dit soort objecten uitrekenen.

Het uiteindelijke doel zal zijn om een werkende Python-module te schrijven die gebaseerd is op abstractere wiskundige stellingen over simplices. Uiteindelijk moet deze module het kunstgalerijprobleem praktisch kunnen oplossen voor een willekeurig aantal dimensies.

2 Kunstgalerijen in twee dimensies

Men heeft veel onderzoek gedaan naar de tweedimensionale versie van het kunstgalerijprobleem. Diverse stellingen laten zich verrassend genoeg echter niet gemakkelijk generaliseren naar meerdere dimensies. In dit hoofdstuk kijken we eerst naar de precieze formulering van het kunstgalerijprobleem. Daarna onderzoeken we twee stellingen over kunstgalerijen en hun bewijzen.

2.1 Definitie en voorbeelden

We volgen de definities uit [3]. Een kunstgalerij is gedefinieerd aan de hand van het begrip *veelhoek*, ook wel *polygoon* genoemd:

Definitie 2.1 (Veelhoek)**.** *Zij $V_1, V_2, ..., V_n$ met $V_i \neq V_j$ voor alle $i \neq j$ een collectie punten in $\mathbb{R}^2$. Dan is een* ***veelhoek*** *de verzameling lijnstukken $V_1V_2, V_2V_3, ..., V_{n-1}V_n, V_nV_1$.*

Merk op dat de lijnstukken elkaar kunnen snijden. Als dit het geval is noemt men de veelhoek *complex*. Veelhoeken die zichzelf niet snijden behoren tot de speciale klasse van *enkelvoudige* veelhoeken. Een eindpunt van een lijnstuk wordt ook wel een *vertex* genoemd (meervoud *vertices*). Er zijn evenveel vertices als lijnstukken in een tweedimensionale veelhoek. Bij het kunstgalerijprobleem zien we een enkelvoudige veelhoek als de "rand"van een kunstgalerij. Om dit preciezer te maken hebben we de stelling van Jordan nodig.

Stelling 2.2 (Jordan)**.** *Zij P een topologische inbedding van de eenheidscirkel S_1 in $\mathbb{R}^2$. Dan bestaat $\mathbb{R}^2 \setminus P$ uit twee samenhangscomponenten, beide met rand P, waarvan er één begrensd is, en de ander onbegrensd.*

De stelling bevestigt wat intuïtief ook al duidelijk is: een lus in het vlak verdeelt het vlak in een binnen- en buitenkant. Het is bovendien niet moeilijk in te zien dat een enkelvoudige veelhoek zelf een inbedding van S_1 is. Ook bij een enkelvoudige veelhoek kunnen we dus spreken van een binnen- en buitenkant. De wiskundige definitie van een kunstgalerij volgt nu op een natuurlijke manier, en is met de stelling van Jordan gerechtvaardigd.

Definitie 2.3 (Kunstgalerij)**.** *Een* ***kunstgalerij*** *is de topologische afsluiting van het begrensde samenhangende gebied omgeven door een enkelvoudige veelhoek.*

Omdat een kunstgalerij volledig wordt bepaald door zijn veelhoek zullen we deze twee begrippen vanaf nu met elkaar identificeren. Als we het dus bijvoorbeeld hebben over "hoekpunten van een kunstgalerij P", dan bedoelen we dus eigenlijk de hoekpunten van de veelhoek die de rand is van P. Hetzelfde geldt voor de "lijnstukken van een kunstgalerij P", waarmee we de lijnstukken van de veelhoek bedoelen die P omgeeft.

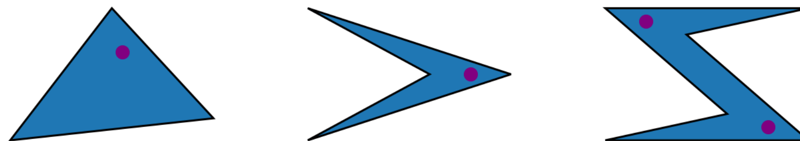
Natuurlijk moeten er ook bewakers of camera's geplaatst worden die de kunstgalerij bewaken. We zullen een *bewaker* zien als een punt in de kunstgalerij. In

een kunstgalerij P "ziet"de bewaker B alle punten X waarvoor geldt dat het lijnstuk $BX \subseteq P$. Merk op dat een bewaker niets kan zien als hij buiten P geplaatst is, en dat hij wel gebieden kan zien als hij op een hoekpunt of op een rand van P geplaatst is.

Een equivalente uitspraak van het klassieke kunstgalerijprobleem werd als volgt door Victor Klee in 1973 geformuleerd:

Gegeven een kunstgalerij. Hoeveel bewakers zijn er dan minimaal nodig om alle punten in de kunstgalerij te bewaken?

Voorbeeld 2.4. *Enkele voorbeelden van kunstgalerijen. De paarse stippen geven aan op welke punten de bewakers geplaatst kunnen worden om de kunstgalerij te bewaken.*



De bovenstaande voorbeelden geven meteen al een paar interessante observaties. In het eerste voorbeeld hebben we een enkelvoudige veelhoek met het minst mogelijke aantal lijnstukken. Het blijkt dat we één bewaker nodig hebben, die bovendien overal in de kunstgalerij geplaatst kan worden. Dit is in het tweede voorbeeld met vier lijnstukken al anders. Daar is ook slechts één bewaker nodig, maar het maakt dit keer wel uit waar we deze bewaker plaatsen. Pas bij zes lijnstukken zijn we in staat om een veelhoek te construeren waarbij er minimaal twee bewakers nodig zijn om de hele ruimte te bewaken.

Van een bepaalde categorie veelhoeken is het zeer gemakkelijk om het kunstgalerijprobleem op te lossen.

Definitie 2.5 (Convexe veelhoek)**.** *Een veelhoek V heet* ***convex*** *als voor elk paar punten x, y binnen de veelhoek geldt dat het lijnstuk $xy \subseteq V$.*

Een convexe veelhoek heeft de eigenschap dat alle interne hoeken kleiner of gelijk aan π zijn. Ook andersom geldt dat elke veelhoek met interne hoeken allen kleiner of gelijk aan π convex is.

Propositie 2.6. *Zij P een kunstgalerij omgeven door een convexe veelhoek. Dan is één bewaker B voldoende om P te bewaken. Bovendien maakt het niet uit waar B in P geplaatst wordt.*

Bewijs. Als X binnen P ligt, dan $BX \subseteq P$ omdat P convex is. □

2.2 De stelling van Chvátal

Men ziet een verband tussen het aantal lijnstukken in de veelhoek en het aantal bewakers dat minimaal nodig is om de bijbehorende kunstgalerij te bewaken. De beroemde stelling van Chvátal geeft een bovengrens voor het minimale aantal bewakers dat nodig is, gegeven het aantal hoekpunten van de veelhoek. Het bewijs is onder andere opgenomen in [3].

Stelling 2.7 (Chvátal)**.** *Zij P een kunstgalerij met n vertices. Dan zijn er nooit meer dan $\lfloor \frac{n}{3} \rfloor$ bewakers nodig om P te bewaken.*

Om deze stelling te bewijzen maken we gebruik van een techniek die *triangulatie* heet. Omdat varianten op deze techniek veel gebruikt worden om andere stellingen over kunstgalerijen mee te bewijzen is het noodzakelijk om dit proces goed te begrijpen.

Definitie 2.8 (Triangulatie)**.** *Een verzameling driehoeken $\mathcal{T}$ is een* ***triangulatie*** *van een kunstgalerij P als:*

- $\bigcup_{T \in \mathcal{T}} T = P$.
- *$T_1 \cap T_2$ is een lijnstuk, een punt of leeg voor alle $T_1, T_2 \in \mathcal{T}$.*

Propositie 2.9. *Zij P omgeven door een enkelvoudige veelhoek met vertices $V_1, V_2, ..., V_n$ en lijnstukken $V_1V_2, V_2V_3, ..., V_nV_1$. Dan bestaat er een triangulatie voor P.*

Bewijs. Inductie naar het aantal vertices n van P. Stel $n = 3$, dan is P een driehoek en dus zelf een triangulatie van P.

Stel nu dat we voor elke veelhoek met $n - 1$ of minder vertices een triangulatie kunnen vinden. Beschouw nu de twee aangrenzende lijnstukken V_1V_2 en V_2V_3 met gemeenschappelijk hoekpunt V_2. Neem zonder verlies van algemeenheid aan dat de interne hoek van V_2 kleiner of gelijk aan π is (permuteer anders de nummering cyclisch). Dan zijn er twee mogelijkheden.

De eerste mogelijkheid is dat het lijnstuk V_1V_3 geheel binnen P ligt en dat V_1V_3 geen ander hoekpunt snijdt. Zij $\mathcal{T}$ de triangulatie van de veelhoek met lijnstukken $V_1V_3, V_3V_4, ..., V_nV_1$. We weten dat een dergelijke triangulatie bestaat omdat deze veelhoek $n - 1$ vertices heeft. Dan is $\mathcal{T} \cup T$ een triangulatie voor P met T de driehoek $V_1V_2V_3$.

De andere mogelijkheid is dat V_1V_3 niet geheel binnen P ligt (of een ander hoekpunt van P snijdt). Dan moet er echter een $t \in [0, 1)$ bestaan zodanig dat het lijnstuk L_t tussen het punt $V_1 + t(V_2 - V_1)$ en het punt $V_3 + t(V_2 - V_3)$ wel in P ligt. Neem de minimale waarde voor een dergelijke t. Dan ligt er ten minste één hoekpunt V_k van P op L_t (anders zouden we t nog kleiner kunnen maken). Het lijnstuk V_kV_2 ligt dan geheel binnen P. We kunnen nu triangulatie $\mathcal{T}_1$ van de veelhoek met lijnsegmenten $V_1V_2, V_2V_k, V_kV_{k+1}, ..., V_nV_1$ en triangulatie $\mathcal{T}_2$ van

de veelhoek met lijnsegmenten $V_2V_3, V_3V_4, ..., V_{k-1}V_k, V_kV_2$ beschouwen. Dan is $\mathcal{T}_1 \cup \mathcal{T}_2$ een triangulatie van P. $\square$

Het handige van triangulatie is dat we het kunstgalerijprobleem kunnen omzetten van een meetkundig probleem naar een grafentheoretisch probleem. Stel P is een kunstgalerij en $\mathcal{T}$ is een triangulatie van P. Zij V de verzameling hoekpunten van P en zij E de collectie ongeordende paren van hoekpunten waartussen zich een zijde van een driehoek $T \in \mathcal{T}$ bevindt. Dan is $G = (V, E)$ een graaf met knopen V en zijden E.

Definitie 2.10. *(n-kleurbaar) Een graaf $G = (V, E)$ heet **n-kleurbaar** als we een verzameling kleuren $C = \{c_1, ..., c_n\}$ en een functie $f : V \to C$ kunnen vinden zodanig dat voor elke zijde bestaande uit het paar punten v_1, v_2 geldt dat $f(v_1) \neq f(v_2)$.*

Propositie 2.11. *Als $G = (V, E)$ een graaf verkregen door triangulatie van een veelhoek P is, dan is G 3-kleurbaar.*

Bewijs. Inductie naar het aantal knopen V. Als $\#V = 3$, dan klopt de uitspraak van de stelling, want er zijn maar drie knopen om te kleuren. Als $\#V > 3$, dan moet er een zijde e tussen knopen v_1 en v_2 in G zijn die niet een lijnstuk in P representeert, anders is G niet verkregen door triangulatie. Er zijn nu twee deelgrafen $G_1 = (V_1, E_1)$ en $G_2 = (V_2, E_2)$ van G te vinden zodanig dat $V_1 \cup V_2 = V$ en $E_1 \cup E_2 = E$ met $V_1 \cap V_2 = \{v_1, v_2\}$ en $E_1 \cap E_2 = \{e\}$. Zowel G_1 als G_2 hebben een kleiner aantal knopen dan G en zijn daarmee volgens de inductiehypothese 3-kleurbaar. Kies in beide grafen de kleuren voor v_1 en v_2 hetzelfde (dit kan altijd door een of twee kleuren met elkaar te verwisselen in de hele graaf). Definieer nu de kleur van een knoop v van G door de kleur van knoop v in G_1 als $v \in V_1$ en door de kleur van knoop v in G_2 als $v \in V_2$. Dan hebben we een 3-kleuring voor G gevonden. $\square$

Propositie 2.12. *Als $G = (V, E)$ een n-kleurbare graaf is, dan is er minstens één kleur die maximaal $\lfloor \frac{\#V}{n} \rfloor$ keer voorkomt.*

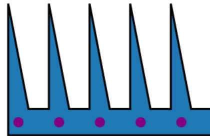
Bewijs. Stel dat alle kleuren meer dan $\lfloor \frac{\#V}{n} \rfloor$ keer zouden voorkomen. Dan geldt dat elke kleur meer dan $\frac{\#V}{n}$ keer voorkomt. Er zijn n kleuren, dus in totaal zouden er dan meer dan $n * \frac{\#V}{n} = \#V$ knopen moeten zijn. Dit is een tegenspraak, dus er moet wel een kleur zijn die maximaal $\lfloor \frac{\#V}{n} \rfloor$ keer voorkomt. $\square$

De stelling van Chvátal werd voor het eerst bewezen door Chvátal zelf. Steve Fisk heeft vervolgens een makkelijkere versie van het bewijs gevonden. Het bewijs werd zodanig elegant bevonden dat het is opgenomen in het beroemde boek *Proofs from THE BOOK* geschreven door Martin Aigner en Günter M. Ziegler. Het bewijs dat hier gegeven wordt is in essentie het bewijs van Fisk:

Bewijs van de stelling van Chvátal. Zij P een kunstgalerij met n vertices. Trianguleer P en kleur de bijbehorende graaf met drie kleuren. Dan moet er een kleur c zijn die maximaal $\lfloor \frac{n}{3} \rfloor$ keer voorkomt. Plaats op elke vertex van P die de kleur c krijgt een bewaker. Dan bewaken deze bewakers de hele kunstgalerij P, omdat elke driehoek in de triangulatie een hoekpunt heeft met de kleur c (anders is het geen 3-kleuring). Omdat een driehoek convex is zijn alle punten in de driehoek zichtbaar vanaf dit hoekpunt. Omdat de unie van alle driehoeken P overdekt is de hele kunstgalerij bewaakt. □

Een logische vervolgvraag is nu of er niet een kleinere ondergrens mogelijk is. We zullen nu een voorbeeld zien waaruit blijkt dat $\lfloor \frac{n}{3} \rfloor$ bewakers soms echt nodig zijn voor elke n.

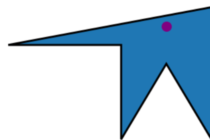
Voorbeeld 2.13. *De volgende kunstgalerij heeft 17 hoekpunten en er zijn minimaal* $\lfloor \frac{17}{3} \rfloor = 5$ *bewakers nodig om de hele galerij te bewaken.*



Het voorbeeld is makkelijk uit te breiden naar elke hoeveelheid vertices.

Bij de stelling van Chvátal is een opmerking te plaatsen. Het bewijs is constructief, dus we hebben een algoritme waarmee we de posities van de bewakers daadwerkelijk kunnen uitrekenen. Dit algoritme laat echter alleen maar bewakers toe die op hoekpunten geplaatst zijn. Dit is een beperking. In sommige gevallen zijn er minder bewakers nodig als we deze toestaan om ook buiten de hoekpunten te staan, zoals in onderstaand voorbeeld.

Voorbeeld 2.14. *Bij deze kunstgalerij is slechts één bewaker voldoende, maar als men deze bewaker op een van de hoekpunten wil plaatsen vindt men geen oplossingen.*



2.3 Orthogonale kunstgalerijen

Tot nu toe hebben we alleen naar kunstgalerijen gekeken met vormen die vooral theoretisch van aard waren. Nu gaan we kijken naar een categorie kunstgalerijen die de werkelijkheid beter benaderen, de zogenoemde *orthogonale kunstgalerijen.*

Definitie 2.15 (Orthogonale Kunstgalerij)**.** *Een kunstgalerij P heet* ***orthogonaal*** *als elke interne hoek tussen twee opeenvolgende lijnstukken van P een veelvoud van $\frac{1}{2}\pi$ is.*

Voor een orthogonale kunstgalerij zijn in het algemeen minder bewakers nodig. Dit heeft te maken met het feit dat we een orthogonale veelhoek kunnen opdelen in (gevulde) convexe vierhoeken in plaats van in driehoeken.

Propositie 2.16. *Zij P een orthogonale kunstgalerij. Dan bestaat er een collectie* ***convexe*** *vierhoeken $\mathcal{Q}$ zodanig dat:*

- $\bigcup_{Q \in \mathcal{Q}} Q = P$.
- *$Q_1 \cap Q_2$ is een vereniging van lijnstukken, een punt of leeg voor alle $Q_1, Q_2 \in \mathcal{Q}$.*

Q wordt ook wel de quadrilateralisatie van P genoemd. Merk op dat het bij deze quadrilateralisatie niet is toegestaan om hoekpunten toe te voegen!

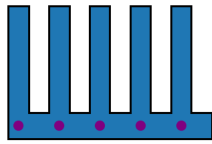
Het bewijs van deze propositie is vrij technisch en werd voor het eerst gegeven door Kahn, Klawe en Kleitman in 1980 [4]. Met deze propositie kunnen we echter gemakkelijk het volgende feit inzien:

Stelling 2.17. *Zij P een orthogonale kunstgalerij met n vertices. Dan zijn er maximaal $\lfloor \frac{n}{4} \rfloor$ bewakers nodig om de hele kunstgalerij te bewaken.*

Bewijs. Deel P eerst op in convexe vierhoeken $\mathcal{Q}$ en bekijk de bijbehorende graaf $G = (V, E)$ waarbij V de vertices van P zijn en E de paren van vertices waartussen een lijnsegment loopt van een $Q \in \mathcal{Q}$, samen met de paren van vertices die tegenover elkaar in een zekere $Q \in \mathcal{Q}$ liggen. Dan is deze graaf planair, ondanks dat de diagonalen van de vierhoeken elkaar lijken te snijden. Neem voor G' de duale graaf van de quadrilateralisatie van P. Dan is G' een boom, want we staan geen gaten toe in de kunstgalerij en we voegen ook geen extra hoekpunten toe, zie ook [3] voor een uitgebreidere analyse. Dit betekent dat we telkens één van de diagonalen van elke vierhoek in G öm P heen kunnen tekenen"zonder dat deze diagonalen elkaar in het vlak snijden. We zien dat G planair is, dus we kunnen dus de beroemde vierkleurenstelling uit de grafentheorie toepassen. Dit impliceert dat er maximaal $\lfloor \frac{n}{4} \rfloor$ bewakers nodig zijn. □

Voor elke n zijn er in bepaalde gevallen ook echt $\lfloor \frac{n}{4} \rfloor$ bewakers nodig en kan men niet met minder volstaan. Om dit te laten zien breiden we het voorbeeld voor gewone kunstgalerijen uit:

Voorbeeld 2.18. *In deze orthogonale kunstgalerij zijn er 22 hoekpunten, maar er zijn $\lfloor \frac{22}{4} \rfloor = 5$ bewakers nodig om de galerij te bewaken. Minder volstaat niet.*



3 Generalisatie naar 3 dimensies

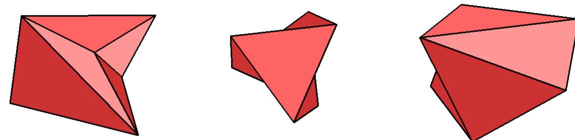
Een kunstgalerij in het dagelijks leven is driedimensionaal. Het lijkt mede hierom erg nuttig om de stellingen uit hoofdstuk 2 te generaliseren. Dit is echter een stuk moeilijker dan het op het eerste gezicht lijkt. Het blijkt dat twee handige eigenschappen van een veelhoek verdwijnen als we ze in drie dimensies beschouwen. Ten eerste is het in het algemeen niet mogelijk een driedimensionaal veelvlak op te delen in convexe delen zonder daarbij punten te moeten toevoegen. Ten tweede is het niet altijd het geval dat bewakers die op *alle* vertices van het veelvlak geplaatst zijn, ook de hele kunstgalerij overzien. Deze feiten bemoeilijken het vinden van een bovengrens en laten zien dat een andere aanpak voor het probleem nodig is voor drie of meer dimensies. In dit hoofdstuk laten we twee illustratieve tegenvoorbeelden zien.

3.1 Het Schönhardt veelvlak

Het equivalent van een veelhoek in drie dimensies wordt een *veelvlak* genoemd. Men kan verschillende definities van een veelvlak aanhouden afhankelijk van de toepassing. Voor nu gebruiken we de definitie dat een veelvlak een topologisch compacte deelruimte van $\mathbb{R}^3$ is, waarvan de rand bestaat uit een eindig aantal veelhoeken die in een vlak in $\mathbb{R}^3$ liggen. We kunnen de veelhoeken zo kiezen dat de doorsnede van twee van deze veelhoeken altijd leeg is, een punt is of een lijnstuk is. Dit is als volgt in te zien. Stel dat de doorsnede van twee veelhoeken tweedimensionaal is, dan moeten de veelhoeken in hetzelfde vlak liggen. We kunnen ze dan samenvoegen tot één veelhoek.

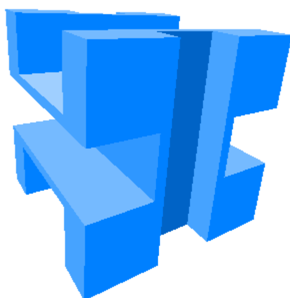
Het lijkt voor de hand te liggen dat elk veelvlak kan worden opgedeeld in convexe veelvlakken, zoals we dat ook in twee dimensies hebben kunnen doen om de stelling van Chvátal te bewijzen. Hier is een tegenvoorbeeld voor [3]. Neem een prisma met als boven- en onderkant een gelijkzijdige driehoek. Deel de rechthoekige zijvlakken van het prisma nu op in twee driehoeken die van elkaar gescheiden worden door een diagonaal van de rechthoek. Zorg ervoor dat de diagonalen geen punten gemeenschappelijk hebben met elkaar en draai vervolgens de bovenkant van het prisma $\frac{1}{6}\pi$ om zijn middelpunt heen. De figuur die men overhoudt heet het **Schönhardt veelvlak** en heeft de bijzondere eigenschap dat we geen tetraëder kunnen vinden met vertices die overeenstemmen met de figuur zelf zodanig dat deze tetraëder geheel binnen de figuur ligt.

in de onderstaande afbeeldingen ziet men het Schönhardt veelvlak vanaf verschillende hoeken bekeken.



3.2 De Octoplex

In een Schönhardt veelvlak zijn bewakers geplaatst op alle vertices voldoende om de hele ruimte te bewaken. Ook het aantal vertices blijkt echter geen bovengrens te zijn voor het minimale aantal bewakers. De *octoplex* blijkt een voorbeeld van een veelvlak waarbij een deel van de ruimte onbewaakt blijft als men op elke vertex een bewaker neerzet. Hieronder is een 3D-model van een octoplex weergegeven.



De formele constructie van dit veelvlak gaat als volgt [5]: Begin met een kubus van $(-10,-10,-10)^T$ naar $(10,10,10)^T$. Verwijder vervolgens deze balken:

- Balk van $(6,10,4)^T$ naar $(-6,-10,10)^T$
- Balk van $(6,10,-4)^T$ naar $(-6,-10,-10)^T$
- Balk van $(10,-10,3)^T$ naar $(-10,-4,-3)^T$
- Balk van $(10,10,3)^T$ naar $(-10,4,-3)^T$
- Balk van $(10,3,10)^T$ naar $(7,-3,-10)^T$
- Balk van $(-10,3,10)^T$ naar $(-7,-3,-10)^T$

Waarbij alle vlakken van deze balken parallel aan een van de vlakken $x = 0$, $y = 0$ en $z = 0$ lopen. Een punt dat niet bewaakt wordt is het punt $(0,0,0)^T$. Dit is in te zien door te checken of elk lijnstuk tussen een hoekpunt van de octoplex en het punt $(0,0,0)^T$ de octoplex snijdt. Met behulp van de simplex-module (verdere uitleg in hoofdstuk 6) is een eenvoudig programma te schrijven dat dit kan doen:

```
# Laad eerst het 3D model, noem het 'octoplex'
for vertex in np.unique(octoplex.vertices, axis = 0):
    found = False
    for simplex in octoplex.polytope:
        if not vertex in simplex:
```

```
        intersection = geometry.simplex_intersection(simplex,
            geometry.Simplex([[0, 0, 0], vertex]))
        if len(intersection) > 0:
            found = intersection
            break
print(found)
```

Deze code geeft een snijpunt van een lijnstuk van een van de hoekpunten naar de oorsprong als er een snijpunt te vinden is, en anders is de output `False`. Een output `False` betekent dus dat het punt $(0, 0, 0)^T$ gezien wordt door een bewaker. Als men deze code echter uitvoert krijgt men een lijst van snijpunten zonder de output `False`. Dat wil zeggen, de oorsprong is niet bewaakt door bewakers op de vertices.

4 Computationele Meetkunde in $\mathbb{R}^n$

Het kunstgalerijprobleem valt onder de *Computationele Meetkunde.* In dit hoofdstuk kijken we naar verschillende begrippen en formules die kunnen helpen bij het bestuderen van een gegeneraliseerde versie van het kunstgalerijprobleem.

4.1 Affiene onafhankelijkheid

Het begrip *affiene onafhankelijkheid* stelt ons in staat om te bepalen of bijvoorbeeld een drietal punten op één lijn ligt.

Definitie 4.1 (Affien onafhankelijk)**.** *Laat $V_0, V_1, ..., V_k \in \mathbb{R}^n$ vectoren zijn. Deze vectoren worden* ***affien onafhankelijk*** *genoemd als de vectoren $V_1 - V_0, V_2 - V_0, ..., V_k - V_0$ lineair onafhankelijk zijn. Een verzameling vectoren heet* ***affien afhankelijk*** *als zij niet affien onafhankelijk zijn.*

Een andere, nuttige karakterisering van affien afhankelijke vectoren is de volgende:

Propositie 4.2. *Een verzameling vectoren $V_0, V_1, ..., V_k \in \mathbb{R}^n$ is affien afhankelijk dan en slechts dan als er $\lambda_0, \lambda_1, ..., \lambda_k \in \mathbb{R}$, niet allen gelijk aan 0 bestaan zodanig dat $\sum_{i=0}^{k} \lambda_i V_i = 0$ en $\sum_{i=0}^{k} \lambda_i = 0$.*

Bewijs. Laat $\lambda_0, \lambda_1, ..., \lambda_k \in \mathbb{R}$ gegeven zijn zo dat $\sum_{i=0}^{k} \lambda_i V_i = 0$ en $\sum_{i=0}^{k} \lambda_i = 0$. Dan $\sum_{i=0}^{k} \lambda_i V_i - \left(\sum_{i=0}^{k} \lambda_i \right) V_0 = 0 \implies \sum_{i=1}^{k} \lambda_i (V_i - V_0) = 0$. Omdat ten minste één van de λ_i ongelijk aan 0 moet zijn, is ook een andere λ_i ongelijk aan 0 omdat $\sum_{i=0}^{k} \lambda_i = 0$. Dit betekent dat er dus een $\lambda_i \neq 0$ bestaat met $i \geq 1$. Dus $V_0, V_1, ..., V_k$ zijn affien afhankelijk.

Veronderstel nu dat $V_0, V_1, ..., V_k$ affien afhankelijk zijn. Dan kunnen we $\alpha_1, ..., \alpha_k \in \mathbb{R}$, niet allen 0 vinden zodanig dat $\alpha_1(V_1 - V_0) + ... + \alpha_k(V_k - V_0) = 0 \implies V_0(-\alpha_1 - ... - \alpha_k) + \alpha_1 V_1 + ... + \alpha_k V_k = 0$. Als we nu $\lambda_i = \alpha_i$ als $i \geq 1$, $\lambda_0 = -\alpha_1 - ... - \alpha_k$ definiëren, dan is ten minste één λ_i ongelijk aan 0, $\sum_{i=0}^{k} \lambda_i V_i = 0$ en $\sum_{i=0}^{k} \lambda_i = 0$. □

Deze definitie is ook de definitie die Grünbaum geeft [2].

4.2 Simplices

Omdat het in hogere dimensies moeilijker wordt om ons een voorstelling te maken van meetkundige objecten zoals punten, lijnen en vlakken zullen we het meer abstracte begrip *simplex (meervoud: simplices)* moeten introduceren.

Definitie 4.3 (Simplex)**.** *Een k-**simplex** $\triangle$ met affien onafhankelijke vertices*

$V_0, V_1, ..., V_k \in \mathbb{R}^n$ is de volgende verzameling punten in $\mathbb{R}^n$:

$$\triangle = \left\{ \sum_{i=0}^{k} \theta_i V_i : \theta_i \in \mathbb{R}_{\geq 0}, \sum_{i=0}^{k} \theta_i = 1 \right\}$$

*Daarnaast wordt een k-simplex $\triangle$ de **standaard k-simplex** genoemd als $V_i = e_i$ voor alle i, met $(e_i)_i = 1$ en $(e_i)_j = 0$ als $j \neq i$. Dan kunnen we $\triangle$ schrijven als:*

$$\triangle = \{x \in \mathbb{R}^k : \sum_{i=0}^{k} x_i = 1, x_i \geq 0\}$$

*Als we het over een **simplex** hebben specificeren we het aantal vertices waar hij uit bestaat niet.*

Voorbeeld 4.4. *Een 0-simplex kennen we als een punt in $\mathbb{R}^n$. Een 1-simplex is een lijnstuk, een 2-simplex is een driehoek en een 3-simplex is een tetraëder, zoals een directe berekening laat zien. Dit rechtvaardigt ook het gebruik van de notatie $\triangle$ voor een simplex.*

Een punt in een simplex kan men ook specificeren door de getallen θ_i te geven in plaats van de Cartesische coördinaten. Deze θ_i noemt men ook wel **barycentrische coördinaten**.

Een belangrijk punt is dat we eisen dat de vertices van een simplex affien onafhankelijk zijn. Dit is absoluut noodzakelijk, zoals de volgende propositie laat zien:

Propositie 4.5. *Laat $V_0, V_1, ..., V_k \in \mathbb{R}^n$ affien afhankelijke vectoren zijn. Laat*

$$\triangle = \{ \sum_{i=0}^{k} \theta_i V_i : \theta_i \in \mathbb{R}_{\geq 0}, \sum_{i=0}^{k} \theta_i = 1\}$$

een deelverzameling van $\mathbb{R}^n$ zijn. Dan zijn de barycentrische coördinaten van een punt $x \in \triangle$ niet uniek.

Merk op dat de $\triangle$ in deze propositie *geen* simplex is zoals we die gedefinieerd hebben door de affiene afhankelijkheid.

Bewijs. Laat $x \in \triangle$ en $V_k - V_0 = \sum_{i=1}^{k-1} \beta_i (V_i - V_0)$, $\beta_i \in \mathbb{R}$. Dan $V_k = V_0 + \sum_{i=1}^{k-1} \beta_i (V_i - V_0) \implies x = (\theta_0 + \theta_k - \theta_k \sum_{i=1}^{k-1} \beta_i) V_0 + \sum_{i=1}^{k-1} (\theta_i + \theta_k \beta_i) V_i$. □

We zullen dadelijk zien dat de barycentrische coördinaten voor een punt x in een simplex wel altijd uniek te bepalen zijn. Voordat we berekeningen uitvoeren met simplices moeten we dus altijd controleren of de vertices niet affien afhankelijk zijn.

Vaak is het handiger om niet de vertices zelf te gebruiken, maar de richtingvectoren ten opzichte van het basispunt V_0. Noteer deze richtingvectoren als

$E_i := V_i - V_0$. We kunnen een simplex uitdrukken in termen van V_0 en de richtingvectoren.

Propositie 4.6. *Zij $\triangle$ een k-simplex met vertices $V_0, V_1, ..., V_k \in \mathbb{R}^n$ en $k \neq 0$. Noteer $E_i := V_i - V_0$. Dan:*

$$\triangle = \{V_0 + \sum_{i=1}^{k} \theta_i E_i : \theta_i \in \mathbb{R}_{\geq 0}, \sum_{i=1}^{k} \theta_i \leq 1\}$$

Deze θ_i stemmen overeen met de barycentrische coördinaten (zonder θ_0).

Bewijs.

$$\begin{aligned}
P &= \sum_{i=0}^{k} \theta_i V_i, & \theta_i \geq 0, \quad & \sum_{i=0}^{k} \theta_i = 1 & \iff \\
P &= V_0 + \sum_{i=0}^{k} \theta_i V_i - V_0 \sum_{i=0}^{k} \theta_i, & \theta_i \geq 0, \quad & \sum_{i=0}^{k} \theta_i = 1 & \iff \\
P &= V_0 + \sum_{i=1}^{k} \theta_i E_i, & \theta_i \geq 0, \quad & \sum_{i=1}^{k} \theta_i + \theta_0 = 1 & \iff \\
P &= V_0 + \sum_{i=1}^{k} \theta_i E_i, & \theta_i \geq 0, \quad & \sum_{i=1}^{k} \theta_i \leq 1 &
\end{aligned}$$

□

Dit geeft nog een belangrijke reden om te eisen voor affiene onafhankelijkheid. Stel namelijk dat $V_0, V_1, ..., V_k \in \mathbb{R}^n$ affien *afhankelijke* vectoren zijn en dat $\triangle = \{V_0 + \sum_{i=1}^{k} \theta_i E_i : \theta_i \in \mathbb{R}_{\geq 0}, \sum_{i=0}^{k} \theta_i \leq 1\}$ (dus $\triangle$ is *geen* simplex), zoals in propositie 4.6. Dan is $\triangle$ getransleerd met $-V_0$, dus de verzameling $\{\sum_{i=1}^{k} \theta_i E_i : \theta_i \in \mathbb{R}_{\geq 0}, \sum_{i=0}^{k} \theta_i \leq 1\}$, een deelverzameling van een vectorruimte met dimensie $< k$ omdat de E_i nu lineair afhankelijk zijn. Met andere woorden, een simplex met affien afhankelijke vectoren zou er plat uitzien ten opzichte van een simplex met hetzelfde aantal affien onafhankelijke vectoren.

Een ander nuttig begrip bij een simplex is een *vlak*:

Definitie 4.7 (vlak van een k-simplex)**.** *Een **l-vlak** van een k-simplex is een deelverzameling van de k-simplex waar $k - l$ barycentrische coördinaten 0 zijn.*

Het is gemakkelijk na te gaan dat l-vlakken van simplices zelf ook simplices zijn.

Voorbeeld 4.8. *De 2-vlakken van een tetraëder zijn de driehoeken die aan het oppervlak van de tetraëder liggen. De 1-vlakken zijn de lijnstukken die de ribben van de tetraëder vormen. De 0-vlakken zijn de vertices van de tetraëder.*

4.3 Punt in een Simplex

Een van de meest voor de hand liggende dingen die we graag zouden willen doen is de vertaalslag maken van Cartesische coördinaten naar barycentrische coördinaten (de andere kant op is een kwestie van vectoren optellen). Dit gaat gemakkelijk als we technieken uit de lineaire algebra toepassen.

Propositie 4.9. *Zij $\triangle$ een k-simplex met vertices $V_0, V_1, ..., V_k \in \mathbb{R}^k$. Noteer $E_i := V_i - V_0$. Dan worden de barycentrische coördinaten van een punt $P \in \mathbb{R}^k$ gegeven door:*

$$(\theta_1, \theta_2, ..., \theta_k)^T = \begin{pmatrix} \vec{E_1} & \vec{E_2} & ... & \vec{E_k} \end{pmatrix}^{-1} (P - V_0)$$

$$\theta_0 = 1 - \sum_{i=1}^{k} \theta_i$$

waarbij $\begin{pmatrix} \vec{E_1} & \vec{E_2} & ... & \vec{E_k} \end{pmatrix}$ de matrix is met als kolommen de vectoren E_i. De pijlen boven de E_i in de formule zijn om duidelijk te maken dat het hier echt om een vierkante matrix gaat.

De formule is ook geldig als P niet in $\triangle$ ligt. Zoals we straks zullen zien geeft deze formule een handige manier om snel te kunnen bepalen of een punt P wel of niet in $\triangle$ ligt. Het bewijs is niet moeilijk en is een kwestie van definities uitschrijven.

Bewijs. Schrijf P eerst als lineaire combinatie van V_0 en E_i. Dit kan altijd, want er zijn k lineair onafhankelijke E_i en P is zelf k-dimensionaal.

$$P = V_0 + \sum_{i=1}^{k} \sigma_i E_i \qquad \Longrightarrow$$

$$P - V_0 = \begin{pmatrix} \vec{E_1} & \vec{E_2} & ... & \vec{E_k} \end{pmatrix} (\theta_1, \theta_2, ..., \theta_k)^T \qquad \Longrightarrow$$

$$(\theta_1, \theta_2, ..., \theta_k)^T = \begin{pmatrix} \vec{E_1} & \vec{E_2} & ... & \vec{E_k} \end{pmatrix}^{-1} (P - V_0)$$

De matrix $\begin{pmatrix} \vec{E_1} & \vec{E_2} & ... & \vec{E_k} \end{pmatrix}$ is altijd inverteerbaar door de lineaire onafhankelijkheid van E_i. Omdat $\sum_{i=0}^{k} \theta_i = 1$ geldt $\theta_0 = 1 - \sum_{i=1}^{k} \theta_i$. □

Door te eisen dat vertices van een simplex altijd affien onafhankelijk moeten zijn hebben we garantie dat elk punt in $\mathbb{R}^k$ te schrijven is in barycentrische coördinaten van *elke* k-simplex in $\mathbb{R}^k$. Dit ligt anders als we de barycentrische coördinaten willen bepalen van een punt $P \in \mathbb{R}^n$ voor een k-simplex in $\mathbb{R}^n$ met $k < n$. In dat geval is het stelsel $P - V_0 = \begin{pmatrix} \vec{E_1} & \vec{E_2} & ... & \vec{E_k} \end{pmatrix} (\theta_1, \theta_2, ..., \theta_k)^T$ overgedetermineerd. We kunnen controleren of er een oplossing is door de rang van

$$\begin{pmatrix} \vec{E_1} & \vec{E_2} & ... & \vec{E_k} \end{pmatrix}$$

te vergelijken met de rang van

$$\left(\vec{E_1} \quad \vec{E_2} \quad ... \quad \vec{E_k} \quad \vec{P} - \vec{V_0}\right).$$

Zijn de beide rangen gelijk, dan is er een oplossing. Als er een oplossing is, dan is deze uniek door de affiene onafhankelijkheid van de vertices. Met technieken als Gauss-eliminatie is deze oplossing te bepalen, maar voor onze doeleinden zullen we (bijna) nooit met een overgedetermineerd stelsel werken.

Bepalen of een punt $P \in \mathbb{R}^n$ in een k-simplex ligt is nu makkelijk door te checken of alle θ_i met $1 \leq i \leq k$ groter of gelijk zijn aan 0, en door te checken of $\sum_{i=1}^{k} \theta_i \leq 1$.

Voor een implementatie in de computer kan het handig zijn om de propositie in een iets andere vorm te schrijven met behulp van de regel van Cramer. We maken hier gebruik van de sgn : $\mathbb{R} \to \{-1, 0, 1\}$ functie, met $\text{sgn}(x) = \frac{x}{|x|}$ als $x \neq 0$ en $\text{sgn}(x) = 0$ als $x = 0$.

Propositie 4.10. *Zij $P \in \mathbb{R}^k$, $\triangle$ een k-simplex met vertices $V_0, V_1, ..., V_k \in \mathbb{R}^k$ en $k \neq 0$. Noteer $E_i := V_i - V_0$ en definieer $D := \det\left(\vec{E_1} \quad ... \quad \vec{E_k}\right)$. Dan ligt P in $\triangle$ dan en slechts dan als:*

- $$\text{sgn}(D) \sum_{i=1}^{k} \det\left(\vec{E_1} \quad ... \quad \vec{E_{i-1}} \quad \vec{P} - \vec{V_0} \quad \vec{E_{i+1}} \quad ... \quad \vec{E_k}\right) \leq |D|$$
- $$\text{sgn}(D) \det\left(\vec{E_1} \quad ... \quad \vec{E_{i-1}} \quad \vec{P} - \vec{V_0} \quad \vec{E_{i+1}} \quad ... \quad \vec{E_k}\right) \geq 0 \quad \forall_{1 \leq i \leq k}$$

Bewijs. De regel van Cramer geeft voor het stelsel

$$P - V_0 = \left(\vec{E_1} \quad \vec{E_2} \quad ... \quad \vec{E_k}\right)(\theta_1, \theta_2, ..., \theta_k)^T$$

de volgende oplossing:

$$\theta_i = \frac{\det\left(\vec{E_1} \quad ... \quad \vec{E_{i-1}} \quad \vec{P} - \vec{V_0} \quad \vec{E_{i+1}} \quad ... \quad \vec{E_k}\right)}{\det\left(\vec{E_1} \quad ... \quad \vec{E_k}\right)}.$$

Invullen van de θ_i in de voorwaarden geeft de formules. □

Voor deze directe rekenmethode zijn geen delingen nodig en daarom kan het voor een computer een kleine tijdswinst opleveren.

4.4 Vlakken in $\mathbb{R}^n$

Voor het kunstgalerijprobleem zullen we moeten uitrekenen of bepaalde punten zich voor of achter een object bevinden ten opzichte van de bewaker. Dit object zal in een vlak liggen in $\mathbb{R}^n$, zoals we later in de precieze definitie van een kunstgalerij zullen zien en wat ook intuïtief duidelijk is in $\mathbb{R}^2$ en $\mathbb{R}^3$. De algemene vraag die we ons zullen stellen is: Gegeven een punt P, een punt Q en een vlak S. Liggen P en Q dan aan dezelfde kant van S of aan tegenovergestelde kanten van S? Laten we de het begrip "vlak"preciezer maken in $\mathbb{R}^n$.

Definitie 4.11 (Vlak)**.** *Een vlak S in $\mathbb{R}^n$ met basisvector $B \in \mathbb{R}^n$ en lineair onafhankelijke richtingvectoren $D_1, ..., D_{n-1} \in \mathbb{R}^n$ is de verzameling:*

$$S = \{B + \sum_{i=1}^{n-1} \alpha_i D_i : \alpha_i \in \mathbb{R}\}$$

Om te bepalen of een lijnstuk PQ (een simplex als $P \neq Q$!) een vlak S met basisvector $B \in \mathbb{R}^n$ en lineair onafhankelijke richtingvectoren $D_1, ..., D_{n-1} \in \mathbb{R}^n$ snijdt, moeten we de vergelijking

$$P + \theta(Q - P) = B + \sum_{i=1}^{n-1} \alpha_i D_i$$

oplossen voor θ. Dit kan alleen op een unieke wijze als $P \neq Q$, dus we nemen dit vanaf nu aan. Als $\theta \in [0, 1]$, dan snijdt PQ het vlak. Met de regel van Cramer verkrijgen we:

$$\theta = \frac{\det\begin{pmatrix}\vec{D_1} & \vec{D_2} & ... & \vec{D_{n-1}} & \vec{P} - \vec{B}\end{pmatrix}}{\det\begin{pmatrix}\vec{D_1} & \vec{D_2} & ... & \vec{D_{n-1}} & \vec{P} - \vec{Q}\end{pmatrix}}$$

Noem $D := \det\begin{pmatrix}\vec{D_1} & \vec{D_2} & ... & \vec{D_{n-1}} & \vec{P} - \vec{Q}\end{pmatrix}$. Als $D = 0$, dan ligt PQ parallel aan S. Merk op dat P en Q dan automatisch aan dezelfde kant van het vlak liggen. Een lijnstuk PQ snijdt S dus als aan de volgende voorwaarden wordt voldaan:

- $D \neq 0 \quad \text{of} \quad P \in S \wedge Q \in S$
- $\text{sgn}(D) \det\begin{pmatrix}\vec{D_1} & \vec{D_2} & ... & \vec{D_{n-1}} & \vec{P} - \vec{B}\end{pmatrix} \geq 0$
- $\text{sgn}(D) \det\begin{pmatrix}\vec{D_1} & \vec{D_2} & ... & \vec{D_{n-1}} & \vec{P} - \vec{B}\end{pmatrix} \leq |D|$

We hebben de volgende propositie over lijnen en vlakken in $\mathbb{R}^n$ aangetoond:

Propositie 4.12. *Zij $P, Q \in \mathbb{R}^n$ en S een vlak met basisvector $B \in \mathbb{R}^n$ en lineair onafhankelijke richtingvectoren $D_1, ..., D_{n-1} \in \mathbb{R}^n$.*

Zij $D := \det\begin{pmatrix}\vec{D_1} & \vec{D_2} & ... & \vec{D_{n-1}} & \vec{P} - \vec{Q}\end{pmatrix}$. Dan liggen P en Q aan tegenovergestelde kanten van S dan en slechts dan als:

- $\text{sgn}(D) \det\begin{pmatrix}\vec{D_1} & \vec{D_2} & ... & \vec{D_{n-1}} & \vec{P} - \vec{B}\end{pmatrix} > 0$
- $\text{sgn}(D) \det\begin{pmatrix}\vec{D_1} & \vec{D_2} & ... & \vec{D_{n-1}} & \vec{P} - \vec{B}\end{pmatrix} < |D|$

Merk op dat het gelijkteken verdwenen is.

$$\text{sgn}(D) \det\begin{pmatrix}\vec{D_1} & ... & \vec{D_{n-1}} & \vec{P} - \vec{B}\end{pmatrix} = 0$$

zou namelijk impliceren dat ofwel $D = 0$, dat wil zeggen dat PQ parallel aan S ligt, ofwel $\theta = 0$, dat wil zeggen dat P op S ligt. Ook zou

$$\text{sgn}(D) \det\begin{pmatrix}\vec{D_1} & ... & \vec{D_{n-1}} & \vec{P} - \vec{B}\end{pmatrix} = |D|$$

betekenen dat $\theta = 1$, dus dat Q op S ligt.

Dit kan ook handig zijn als we willen bepalen of twee punten aan dezelfde of aan de tegenovergestelde kant van een $n-1$-simplex in $\mathbb{R}^n$ liggen. We kunnen ook bepalen of de ene simplex achter de andere ligt ten opzichte van een bepaald punt. Deze methode werd in de beginperiode van 3D computer graphics gebruikt om te bepalen in welke volgorde objecten op het scherm getekend moesten worden.

Een gerelateerd probleem is het vinden van het snijpunt van een lijnstuk met een $n-1$-simplex in $\mathbb{R}^n$. Ook dit is relevant voor het kunstgalerijprobleem. Als we een bewaker hebben op punt P die in een richting Q kijkt, willen we weten of er simplices zijn die een snijpunt met (het verlengde van) PQ hebben. Dit is overigens grofweg de methode die tegenwoordig voor $n = 3$ wordt gebruikt in 3D computer graphics. Voor elke pixel in het scherm berekent de computer of en waar de lichtstraal de driehoeken in $\mathbb{R}^3$ snijdt. Deze methode wordt ook wel *ray casting* of *ray shooting* genoemd.

Dit probleem is nagenoeg hetzelfde, met als uitzondering dat we in plaats van de basisvector B werken met de vector V_0. In plaats van richtingvectoren D_i maken we gebruik van richtingvectoren E_i. Ook moeten we nu de coëfficiënten α_i uitrekenen, die overeenstemmen met de barycentrische coördinaten, en checken of die voldoen aan de voorwaarden van een simplex. Samengevat geldt de volgende propositie:

Propositie 4.13. *Zij $\triangle$ een $n-1$-simplex in $\mathbb{R}^n$ met vertices $V_0, ..., V_{n-1}$. Noteer $E_i := V_i - V_0$. Zij L een 1-simplex met vertices P en Q. Dan kan de doorsnede $L \cap \triangle$ worden bepaald met behulp van de barycentrische coördinaten $\theta_1, ..., \theta_{n-1}$ van $\triangle$ of met behulp van het barycentrische coördinaat α van L. Deze zijn gegeven door:*

$$(\theta_1, ..., \theta_{n-1}, \alpha)^T = \begin{pmatrix} \vec{E_1} & ... & \vec{E_{n-1}} & \vec{P} - \vec{Q} \end{pmatrix}^{-1} (P - V_0)$$

als $\det \begin{pmatrix} \vec{E_1} & ... & \vec{E_{n-1}} & \vec{P} - \vec{Q} \end{pmatrix} \neq 0$

In sommige situaties is het handig om een punt loodrecht op een vlak te projecteren.

Propositie 4.14. *Zij $P \in \mathbb{R}^n$ en S een vlak met basisvector $B \in \mathbb{R}^n$ en lineair onafhankelijke richtingvectoren $D_1, ..., D_{n-1} \in \mathbb{R}^n$. Dan:*

$$\text{proj}_S(P) = B + \sum_{i=1}^{n-1} \alpha_i D_i$$

Waarbij

$$(\alpha_1, ..., \alpha_{n-1})^T = M^{-1}(< P - B, D_1 >, ..., < P - B, D_{n-1} >)$$

$$M = \begin{pmatrix} < D_1, D_1 > & < D_2, D_1 > & \dots & < D_{n-1}, D_1 > \\ < D_1, D_2 > & < D_2, D_2 > & \dots & < D_{n-1}, D_2 > \\ \vdots & \vdots & \ddots & \vdots \\ < D_1, D_{n-1} > & < D_2, D_{n-1} > & \dots & < D_{n-1}, D_{n-1} > \end{pmatrix}$$

Bovendien geldt $\det M \neq 0$.

Bewijs. Zij $U \in \mathbb{R}^n$ zodanig dat U orthogonaal is met alle richtingvectoren D_i. Dan is P te schrijven als:

$$P = B + U + \sum_{i=1}^{n-1} \alpha_i D_i$$

Inproduct nemen met D_j levert:

$$< P - B, D_j >= \sum_{i=1}^{n-1} \alpha_i < D_i, D_j >$$

en dit is precies het stelsel dat we zoeken. Samen met het feit dat $\text{proj}_S(P) = P - U$ levert dit de formule.

We moeten nu slechts aantonen dat de determinant van de matrix M ongelijk aan 0 is. Merk op dat als $A = \left(\vec{D_1}, ..., \vec{D_{n-1}}\right)$, dan $M = AA^T$, dus $\det M = \det A \det A^T \neq 0$ omdat $\det A \neq 0$ aangezien de D_i lineair onafhankelijk zijn. □

4.5 Doorsnede van Twee Simplices

Een beroemd probleem in de computationele meetkunde is het bepalen of (en waar) twee willekeurige collecties van simplices elkaar snijden. Dit wordt ook wel het *Collision detection problem* genoemd. De toepassing ligt vooral in het maken van natuurkundige simulaties en in de robotica, maar ook voor het kunstgalerijprobleem is dit interessant. Stel dat we voor elke bewaker een gebied bestaande uit een collectie simplices hebben uitgerekend dat de bewaker niet kan zien. Als we nu kunnen laten zien dat de doorsnede van deze collecties simplices leeg is, dan weten we dat de bewakers de hele kunstgalerij bewaken.

Zij $\triangle_1$ en $\triangle_2$ simplices. $\triangle_1$ en $\triangle_2$ zijn beide convex, dus ook hun doorsnede $U := \triangle_1 \cap \triangle_2$ moet convex zijn. We willen uiteindelijk weten hoe we U precies kunnen beschrijven. Hiervoor is het begrip *convex omhulsel* handig.

Definitie 4.15 (Convex Omhulsel)**.** *Zij* $X = \{x_1, x_2, ..., x_k\} \in \mathbb{R}^n$ *een verzameling punten. Dan is het* ***Convex Omhulsel*** *van* X *de vereniging van alle mogelijke simplices met vertices in* X*.*

Propositie 4.16. *De volgende uitspraken zijn equivalent:*

1. conv$X$ *is het convex omhulsel van* X.
2. conv$X$ *is de kleinste convexe verzameling die* X *omvat.*
3. conv$X$ *is de doorsnede van alle convexe verzamelingen die* X *omvatten.*

Bewijsschets. (2) $\iff$ (3) is niet moeilijk in te zien. Een opmerking is hier wel dat we inderdaad mogen spreken van een kleinste convexe verzameling die X omvat, omdat deze precies gelijk is aan de in (3) beschreven verzameling. Een doorsnede tussen twee convexe verzamelingen is namelijk opnieuw convex. (3) $\iff$ (1) is moeilijker. Het idee is om na te gaan dat conv$X$ een *convexe combinatie* is van punten uit X. Een convexe combinatie is hetzelfde als een simplex, met als enige verschil dat we niet eisen dat de punten affien onafhankelijk zijn. Vervolgens laat de stelling van Carathéodory zien dat in dit geval een convexe combinatie van punten uit X een convexe combinatie van $n+1$ punten uit X impliceert. Dit geeft onze definitie van een convex omhulsel terug. Zie voor een uitgebreidere uitleg [7]. □

Stel we hebben twee simplices $\triangle_1$ en $\triangle_2$ in $\mathbb{R}^n$. Kies nu voor elke simplex respectievelijk een k-vlak en een l-vlak door vertices $V_0, V_1, ..., V_k$ en $U_0, U_1, ..., U_l$ uit de verzameling vertices van $\triangle_1$ en $\triangle_2$ te selecteren. Noem deze vlakken $\triangle_1'$ en $\triangle_2'$. In sommige speciale gevallen bestaat de doorsnede $\triangle_1' \cap \triangle_2'$ uit één punt. Wat zijn de voorwaarden waaraan deze speciale gevallen moeten voldoen?

Noteer

$$\begin{aligned}
&\triangle_1' = \{V_0 + \sum_{i=1}^{k} \theta_i E_i \quad : \quad \forall i \quad \theta_i \geq 0, \quad \sum_{i=1}^{k} \theta_i \leq 1\} \\
&E_i = V_i - V_0 \\
&\triangle_2' = \{U_0 + \sum_{i=1}^{l} \sigma_i F_i \quad : \quad \forall i \quad \sigma_i \geq 0, \quad \sum_{i=1}^{l} \sigma_i \leq 1\} \\
&F_i = U_i - U_0
\end{aligned}$$

Als $\triangle_1' \cap \triangle_2'$ uit één punt bestaat is het volgende stelsel vergelijkingen in ieder geval uniek oplosbaar, hoewel aan de voorwaarden voor θ en σ niet noodzakelijk voldaan is:

$$V_0 + \sum_{i=1}^{k} \theta_i E_i = U_0 + \sum_{i=1}^{l} \sigma_i F_i$$

Uit de lineaire algebra is bekend dat het voldoende is om na te gaan of het stelsel

$$\left(\vec{E}_1 \quad ... \quad \vec{E}_k \quad -\vec{F}_1 \quad ... \quad -\vec{F}_l\right)\left(\theta_1 \quad ... \quad \theta_k \quad \sigma_1 \quad ... \quad \sigma_l\right)^T = U_0 - V_0$$

een unieke oplossing heeft. In de meeste praktische gevallen zal het voorkomen dat de kolommen in de matrix altijd lineair onafhankelijk zijn als $k + l \leq n$. In dat geval heeft het stelsel een unieke oplossing dan en slechts dan als $k + l = n$.

We kunnen nu de doorsnede van de twee simplices $\triangle_1$ en $\triangle_2$ expliciet uitrekenen:

Stelling 4.17. *Zij $\triangle_1, \triangle_2$ simplices in $\mathbb{R}^n$. Dan is $\triangle_1 \cap \triangle_2$ het convex omhulsel van de volgende punten:*

$$X := \{V_0 + \sum_{i=1}^{k} \theta_i E_i \quad :$$

θ_i behoren tot de oplossing van

$$\begin{pmatrix} \vec{E}_1 & \dots & \vec{E}_k & -\vec{F}_1 & \dots & -\vec{F}_l \end{pmatrix} \begin{pmatrix} \theta_1 & \dots & \theta_k & \sigma_1 & \dots & \sigma_l \end{pmatrix}^T = U_0 - V_0$$

waarbij deze oplossing **uniek** is, met

$$\theta_i \geq 0, \quad \sigma_i \geq 0, \quad \sum_{i=1}^{k} \theta_i \leq 1, \quad \sum_{i=1}^{l} \sigma_i \leq 1$$

$V_0, ..., V_k$ zijn elementen uit de verzameling vertices van $\triangle_1$

$U_0, ..., U_l$ zijn elementen uit de verzameling vertices van $\triangle_2\}$

Met notatie $E_i = V_i - V_0$ en $F_i = U_i - U_0$ als gebruikelijk.

Bewijs. Het feit dat $\text{conv}X \subseteq \triangle_1 \cap \triangle_2$ volgt uit het feit dat $\triangle_1 \cap \triangle_2$ convex is, dat elk punt uit X in $\triangle_1 \cap \triangle_2$ ligt en uit de voorgaande propositie.

Het is verder duidelijk dat $\triangle_1 \cap \triangle_2$ een convex omhulsel van eindig veel punten moet zijn. De posities van de vertices van dit convex omhulsel zijn oplossingen van de vergelijking

$$V'_0 + \sum_{i=1}^{k'} \alpha_i E'_i = U'_0 + \sum_{i=1}^{l'} \beta_i F'_i$$

$$\alpha_i \geq 0, \quad \beta_i \geq 0, \quad \sum_{i=1}^{k'} \alpha_i \leq 1, \quad \sum_{i=1}^{l'} \beta_i \leq 1$$

waarvoor dit stelsel unieke oplossingen heeft met $V'_0, ..., V'_{k'}$ de vertices van $\triangle_1$, $U'_0, ..., U'_{l'}$ de vertices van $\triangle_2$, $E'_i = V'_i - V'_0$ en $F'_i = U'_i - U'_0$. Zoals we net gezien hebben heeft deze vergelijking een unieke oplossing als aan de voorwaarden gesteld aan X voldaan is. $\square$

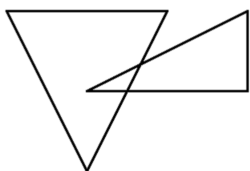
Voor $\mathbb{R}^3$ geldt dus het volgende, expliciet opgeschreven:

Gevolg 4.18. *Zij $\triangle_1$ en $\triangle_2$ simplices in $\mathbb{R}^3$ met vertices $V_0, ..., V_k$ en $U_0, ..., U_l$ met $k + l > 2$ zodanig dat elke collectie van drie vectoren $V_i - V_0$ of $U_i - U_0$*

lineair onafhankelijk is, dan bestaat $\triangle_1 \cap \triangle_2$ uit het convex omhulsel van de volgende punten:

- *Punten U_i die in $\triangle_1$ liggen*
- *Punten V_i die in $\triangle_2$ liggen*
- *Snijpunten van 1-vlakken (lijnstukken) van $\triangle_2$ met 2-vlakken (driehoeken) van $\triangle_1$*
- *Snijpunten van 1-vlakken van $\triangle_1$ met 2-vlakken van $\triangle_2$*

De voorwaarde dat elke collectie van drie vectoren $V_i - V_0$ of $U_i - U_0$ lineair onafhankelijk moet zijn is om de matrix van stelling 4.17 inverteerbaar te maken. Bekijk ter illustratie het volgende voorbeeld: stel dat $\triangle_1$ bestaat uit de vertices $(0,-1,0)^T, (-1,1,0)^T$ en $(1,1,0)^T$ en dat $\triangle_2$ bestaat uit de vertices $(0,0,0)^T, (2,0,0)^T$ en $(2,1,0)^T$, zie onderstaande afbeelding.



De doorsnede van $\triangle_1 \cap \triangle_2$ bestaat nu uit het convex omhulsel van onder andere punten die het snijpunt van twee lijnstukken zijn. Er is bovendien geen snij*punt* meer tussen $\triangle_1$ en een lijnstuk van $\triangle_2$ vanuit de oorsprong. Om dit soort gevallen uit te sluiten eisen we dus dat elke collectie van drie vectoren $V_i - V_0$ of $U_i - U_0$ lineair onafhankelijk moet zijn.

Men berekent $\triangle_1 \cap \triangle_2$ in $\mathbb{R}^3$ dus door elke combinatie van bovenstaande vier punten uit te proberen. Al deze berekeningen kunnen worden gedaan met technieken uit de voorgaande resultaten.

Soms zijn we niet geïnteresseerd in de precieze vorm van $\triangle_1 \cap \triangle_2$, maar willen we weten of de doorsnede überhaupt niet leeg is. Het helpt dan om het probleem om te schrijven naar een algemene matrixvorm [6]:

Propositie 4.19. *Zij $\triangle_1$ een simplex met vertices $V_0, ..., V_k$ en $\triangle_2$ een simplex met vertices $U_0, ..., U_l$ in $\mathbb{R}^n$. Dan is de doorsnede $\triangle_1 \cap \triangle_2 \neq \emptyset$ dan en slechts*

dan als:

$$\begin{pmatrix} (V_0)_1 & (V_1)_1 & ... & (V_k)_1 & -(U_0)_1 & -(U_1)_1 & ... & -(U_l)_1 \\ (V_0)_2 & (V_1)_2 & ... & (V_k)_2 & -(U_0)_2 & -(U_1)_2 & ... & -(U_l)_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ (V_0)_n & (V_1)_n & ... & (V_k)_n & -(U_0)_n & -(U_1)_n & ... & -(U_l)_n \\ 1 & 1 & ... & 1 & -1 & -1 & ... & -1 \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_k \\ \beta_0 \\ \beta_1 \\ \vdots \\ \beta_l \end{pmatrix} = 0$$

een niet-triviale oplossing heeft met $\alpha_i \geq 0$ *en* $\beta_i \geq 0$.

Bewijs. Het bewijs is simpelweg het uitschrijven van de vergelijkingen

$$\sum_{i=0}^{k} \alpha_i V_i = \sum_{i=0}^{l} \beta_i U_i$$
$$\sum_{i=0}^{k} \alpha_i = \sum_{i=0}^{l} \beta_i$$

met als opmerking dat

$$\sum_{i=0}^{k} \alpha_i = \sum_{i=0}^{l} \beta_i$$

een niet-triviale oplossing heeft, equivalent is met

$$\sum_{i=0}^{k} \alpha_i = \sum_{i=0}^{l} \beta_i = 1$$

omdat elke niet-triviale oplossing van $\sum_{i=0}^{k} \alpha_i V_i = \sum_{i=0}^{l} \beta_i U_i$ met $\sum_{i=0}^{k} \alpha_i = \sum_{i=0}^{l} \beta_i$ aanleiding geeft tot een oplossing $\sum_{i=0}^{k} \mu_i V_i = \sum_{i=0}^{l} \nu_i U_i$ met $\mu_i = \frac{\alpha_i}{\sum_{i=0}^{k} \alpha_i}$ en $\nu_i = \frac{\beta_i}{\sum_{i=0}^{l} \beta_i}$ waarbij $\sum_{i=0}^{k} \mu_i = 1$ en $\sum_{i=0}^{l} \nu_i = 1$. □

De vertaling van het probleem naar de vorm

$$Av = 0$$
$$v \geq 0$$

met A een matrix, zorgt ervoor dat we dit met lineair programmeren kunnen oplossen. Zie voor een voorbeeld van een methode ook [6].

5 Kunstgalerijen in $\mathbb{R}^n$

Met een goed begrip van simplices en met de formules die ons in staat stellen om snijpunten van deze simplices met andere objecten te vinden kunnen we formeler naar een meerdimensionale kunstgalerij kijken.

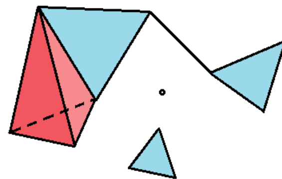
5.1 Polytoop als Kunstgalerij

Voor de generalisatie van het begrip *veelhoek* zijn er meerdere definities gangbaar, afhankelijk van de toepassing. Over het algemeen wordt een meerdimensionale veelhoek een *polytoop* genoemd. Er zijn vele verschillende definities voor een polytoop gangbaar, maar de volgende definitie blijkt geschikt voor het kunstgalerijprobleem en voor de Python-module die we in hoofdstuk 6 zullen behandelen.

Definitie 5.1 (Polytoop)**.** *Een* ***polytoop*** *in* $\mathbb{R}^n$ *is een eindige verzameling simplices waarvoor geldt dat de doorsnede van twee simplices uit deze verzameling óf leeg is, óf bestaat uit een* k*-vlak (definitie 4.7) van beide simplices.*

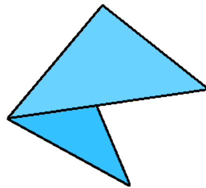
Deze definitie is gerelateerd aan de gangbare definitie van een *convexe polytoop*, het convex omhulsel van een eindig aantal punten in $\mathbb{R}^n$ in die zin dat een vlak van een convexe polytoop doorsneden met een ander vlak van diezelfde polytoop opnieuw een vlak van de polytoop is. Zie ook [2].

Voorbeeld 5.2. *Een voorbeeld van een toegestane polytoop in* $\mathbb{R}^3$ *is de volgende:*



Hierbij is in het rood een tetraëder aangegeven, en in het blauw driehoeken. Deze simplices hoeven niet aan elkaar vast te zitten, dus bijvoorbeeld losse punten of driehoeken zijn toegestaan.

Voorbeeld 5.3. *Een voorbeeld van een object dat geen polytoop in* $\mathbb{R}^2$ *is:*



De doorsnede van de grote driehoek met de kleine is niet-leeg en bestaat uit het bovenste lijnstuk van de kleine driehoek. Aan de voorwaarde dat de doorsnede

moet bestaan uit een k-vlak van beide simplices is niet voldaan, aangezien dit lijnstuk geen vlak is van de grote driehoek. Om hier een geldige polytoop van de maken zal men de grote driehoek moeten opsplitsen door een lijnstuk van de bovenste vertex naar het bovenste hoekpunt van de kleine driehoek toe te voegen.

Definitie 5.4 (Kunstgalerij)**.** *Een* ***kunstgalerij*** *in* $\mathbb{R}^n$ *is een polytoop, bestaande uit n-1-simplices, die de rand vormt van een compacte deelverzameling van* $\mathbb{R}^n$*.*

Het is makkelijk na te gaan dat de oorspronkelijke definitie voor een 2D kunstgalerij hier aan voldoet. We kunnen over een kunstgalerij nadenken als een betegeling van een oppervlak in $\mathbb{R}^n$. Dit is ook hoe 3D modellen in een computer gemaakt kunnen worden. Deze 3D modellen bestaan uit niets anders dan een verzameling driehoeken die mogelijk met hun randen aan elkaar vast zitten. Zie voor meer informatie over de betegeling van een 3D oppervlak [1].

5.2 Schaduwgebieden in een Kunstgalerij

Neem aan dat een bewaker B in een n-dimensionale kunstgalerij staat. We willen nu weten welke gebieden de bewaker allemaal *niet* kan zien. Laten we eerst naar één simplex uit de polytoop kijken. Deze simplex heeft $n-1$ vertices. Stel nu dat B affien onafhankelijk is van deze vertices (anders liggen B en de vertices in hetzelfde hypervlak in $\mathbb{R}^n$ en blokkeert deze simplex dus niet het zicht van de bewaker). Dan kunnen we een gebied omschrijven dat de "schaduw"gaat vormen van deze simplex:

Definitie 5.5 (Schaduw)**.** *Zij* $\triangle$ *een* $n-1$ *simplex in* $\mathbb{R}^n$ *met vertices* $V_0, ..., V_{n-1}$*. Zij* $B \in \mathbb{R}^n$ *affien onafhankelijk van* $V_0, ..., V_{n-1}$ *en noteer* $E_i := V_{i-1} - B$*. Dan is de* ***schaduw*** *van* $\triangle$ *ten opzichte van* B *gegeven door:*

$$S_B(\triangle) := \left\{ B + \sum_{i=1}^{n} \alpha_i E_i \quad : \quad \alpha_i \geq 0, \quad \sum_{i=1}^{n} \alpha_i > 1 \right\}$$

Een schaduwgebied $S_B(\triangle)$ lijkt erg op het begrip van een simplex met als enige verschil dat de som van barycentrische coördinaten *groter* moet zijn dan 1 in plaats van kleiner dan of gelijk aan 1. Hier zijn dus ook de formules voor simplices te gebruiken, zij het in iets aangepaste vorm met betrekking tot die voorwaarde.

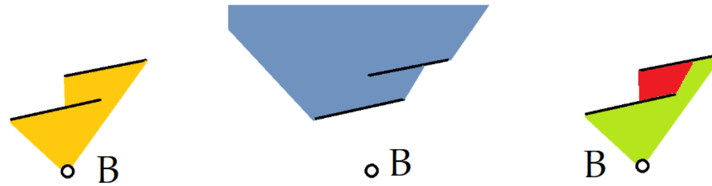
Een schaduwgebied van een simplex ten opzichte van een bewaker is convex, zoals een eenvoudige coördinatentransformatie naar E_i laat zien (een convex gebied dat lineair getransformeerd wordt blijft convex).

De tegenhanger van een schaduwgebied is een *lichtgebied*.

Definitie 5.6 (Lichtgebied)**.** *Een* ***lichtgebied*** *van een* $n-1$*-simplex* $\triangle$ *in* $\mathbb{R}^n$ *ten opzichte van een bewaker* $B \in \mathbb{R}^n$ *is de simplex verkregen door* B *als vertex aan* $\triangle$ *toe te voegen.*

Dit is natuurlijk alleen gedefinieerd als de hoekpunten van dit lichtgebied affien onafhankelijk zijn.

De wisselwerking tussen licht- en schaduwgebieden is belangrijk. In de eerste van de onderstaande drie afbeeldingen is de vereniging van lichtgebieden ten opzichte van B van de twee simplices weergegeven en in de tweede afbeelding de vereniging van schaduwgebieden. Het gebied dat B kan bewaken is in de derde afbeelding in het groen aangegeven. Dit gebied is de vereniging van lichtgebieden zonder de doorsnede van de lichtgebieden met de schaduwgebieden.:



We zien dat we hiermee een systematische methode hebben gevonden om de gebieden uit te rekenen die een bepaalde bewaker kan zien. Een toepassing hiervan kan bijvoorbeeld zijn om uit te rekenen of een bepaalde configuratie van bewakers de hele kunstgalerij bewaakt.

Een algoritme dat voor elke kunstgalerij P een configuratie van bewakers geeft die de hele kunstgalerij bewaken kan als volgt recursief werken met behulp van de schaduwgebieden:

1. Neem $P' = P$.
2. Plaats een bewaker in P' en reken de vereniging $\mathcal{S}$ van alle schaduwgebieden van P' ten opzichte van deze bewaker uit.
3. Reken $D := \mathcal{S} \cap P$ uit.
4. Als D leeg is, stop het algoritme. Neem anders $P' = D$ en begin opnieuw bij stap 2.

Een probleem met dit algoritme is stap 3. We zouden P graag willen opdelen in simplices om dit uit te rekenen, maar deze stap is niet triviaal, zoals we in hoofdstuk 3 al zagen. In plaats van dat we de doorsnede met de kunstgalerij uitrekenen, zouden we ook de doorsnede tussen de schaduw- en lichtgebieden kunnen bepalen. Het probleem is dat deze gebieden voor een deel buiten de kunstgalerij kunnen liggen. We zouden dit op kunnen lossen door simplices van de rand van de kunstgalerij die dit gebied snijden toe te voegen aan D. Mogelijk komt het algoritme met bewakers die buiten de kunstgalerij staan, maar dit is achteraf te controleren. Verder is een interessante vraag voor verder onderzoek of er kunstgalerijen mogelijk zijn waarvoor geldt dat het algoritme oneindig lang doorgaat met bewakers toevoegen.

6 Implementatie in de Computer

De stellingen in de vorige hoofdstukken suggereren een grote praktische toepasbaarheid. In deze sectie kijken we naar een Python-module voor simplices, en hoe we deze module praktisch kunnen gebruiken om het kunstgalerijprobleem in de computer te zetten.

6.1 Simplex-module

Om de toepassingen van de in sectie 4 beschreven algoritmen algemeen te houden is het handig om deze in een module te stoppen. Deze Simplex-module, genaamd *geometry.py*, bestaat uit een klasse van simplices met enkele methodes, en een aantal functies.

6.1.1 Simplex Klasse

De Simplex objecten hebben de volgende eigenschappen:

- `matrix`
- `space`
- `dimension`
- `color`

De `matrix` geeft alle punten van de simplex. De matrix moet worden opgegeven bij initialisatie van het object. De `space` eigenschap is de dimensie van de vectorruimte en wordt berekend aan de hand van de opgegeven matrix. De `dimension` geeft het aantal vertices in de simplex. Tot slot kan met de `color` een eventuele extra eigenschap aan de simplex worden toegekend zoals bijvoorbeeld kleur, of dat de simplex onderdeel is van een bepaald 3D-model.

De belangrijkste methodes zijn:

1. `copy`, die de simplex kopiëert,
2. `remove_duplicates`, die dubbele punten verwijdert,
3. `merge`, die twee simplices samenvoegt tot een nieuwe simplex met dezelfde vertices (controleert niet op affiene afhankelijkheid),
4. `affine_independent`, die controleert of de punten in de simplex affien onafhankelijk zijn,
5. `translate`, die de simplex met een bepaalde vector transleert,
6. `transform`, die alle vertices in de simplex lineair transformeert en
7. `direction_vectors`, die de richtingvectoren van een simplex geeft.

6.1.2 Functies

Elk algoritme beschreven in sectie 4 heeft een bijbehorende functie in de module.

1. `point_in_simplex(point, simplex)`, return: boolean. Geeft aan of `point` in `simplex` ligt.

2. `points_opposite(P, Q, base, directions)`, return: boolean. Geeft aan of `P` en `Q` aan tegenovergestelde kanten van het vlak met basisvector `base` en richtingvectoren `directions` liggen.

3. `simplex_intersection(simplex1, simplex2)`, return: list. Geeft de punten waarvan het convex omhulsel de doorsnede is van `simplex1` met `simplex2`.

4. `simplex_S_intersection(simplex, S, base)`, return: list. Geeft de punten waarvan het convex omhulsel de doornede is van `simplex` en het gebied buiten `S` waarvan de barycentrische coördinaten zonder `S[base]` opgeteld groter zijn dan 1. Dat wil zeggen, het schaduwgebied van `S` zonder `S[base]` ten opzichte van `S[base]`.

5. `simplex_intersection_bool(simplex1, simplex2)`, return: boolean. Geeft aan of de doorsnede van `simplex1` met `simplex2` leeg is of niet.

6. `projection_on_plane(point, base, directions)`, return: array. Geeft de loodrechte projectie van `point` op het vlak met basisvector `base` en richtingvectoren `directions`.

7. `line_plane_intersection(P, Q, base, directions)`, return: array. Geeft het snijpunt van het lijnstuk tussen `P` en `Q` en het vlak met basisvector `base` en richtingvectoren `directions`. Returnt `None` als er geen snijpunt is.

8. `line_plane_barycentric(P, Q, base, directions)`, return: array. Geeft het snijpunt van het lijnstuk tussen `P` en `Q` en het vlak met basisvector `base` en richtingvectoren `directions`in barycentrische coördinaten. Returnt `None` als er geen snijpunt is.

9. `R3_hull_to_polytope(hull)`, return: list. Geeft een lijst met simplices die de rand van het convex omhulsel van `hull` vormen. Werkt slechts in 3 dimensies.

6.2 Visualisatie

In de computer is het alleen mogelijk om een goede visualisatie van een kunstgalerij te maken in 3 dimensies of minder. De voorbeelden die we hier zullen gebruiken zullen dan ook van toepassing zijn op drie dimensies.

6.2.1 Modules

Om een 3D visualisatie te maken zijn een aantal extra stappen nodig. Een computerscherm is immers tweedimensionaal. We zullen dus een projectie van de 3D wereld moeten maken. In vrijwel alle praktische toepassingen wordt hiervoor de grafische processor (GPU) gebruikt. Om die gemakkelijk aan te kunnen spreken maken we gebruik van de module *PyOpenGL*. Deze module stelt ons in staat om OpenGL te gebruiken in Python. Hierdoor is de CPU niet meer nodig om van elke pixel de kleur uit te rekenen (bijvoorbeeld met een ray shooting methode). Dit versnelt het programma aanzienlijk.

Deze significante tijdswinst maakt het ook mogelijk om real-time de positie en richting van de camera te veranderen met behulp van toetsenbord-input. De toetsaanslagen houden we bij met de module *PyGame*, een module ontwikkeld voor het maken van games, maar ook prima te gebruiken voor onze doeleinden.

6.2.2 3D-modellen

De eerste stap is het maken van een geschikt 3D model dat een kunstgalerij representeert. Er is geen standaard bestandsformaat voor een 3D model, maar we hebben in elk geval een kort programma nodig dat een dergelijk bestand kan lezen en de data kan omzetten in een structuur die geschikt is voor PyOpenGL en de Simplex-module. Dat wil zeggen dat het 3D model moet bestaan uit een collectie driehoeken, die elk nog mogen beschikken over kleur, doorzichtigheid en textuurcoördinaten. Deze driehoeken kunnen door

```
glColor4fv(color)
glTexCoord2fv(uv)
glVertex3fv(vertex)
```

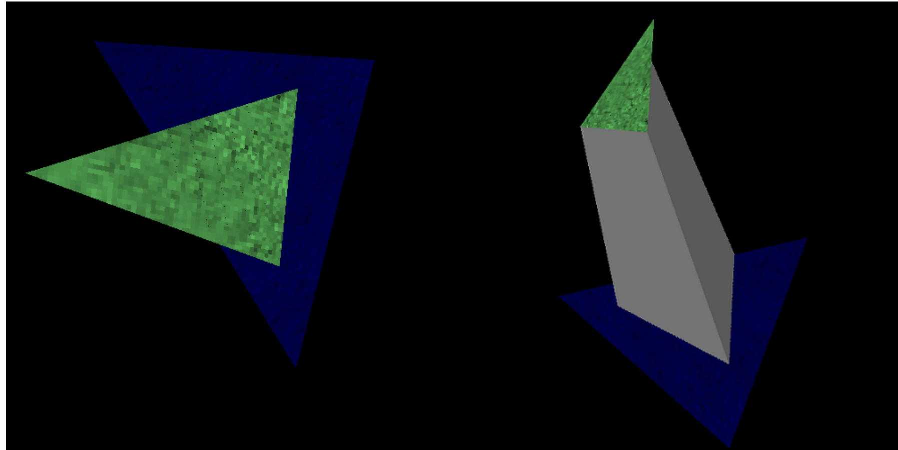
op het scherm weergegeven worden (voor elke driehoek drie van deze aanroepen). Verder moet het 3D model nog worden omgezet naar een collectie van simplex-objecten om er later berekeningen mee uit te voeren.

6.2.3 Camera

Om het 3D model zichtbaar te maken zullen we alleen nog de camera moeten instellen. Dit kan met de functies `gluLookAt` en `gluPerspective`. `gluPerspective` stelt de kijkhoek in, de minimale en maximale afstand die objecten tot de camera moeten hebben om op het scherm getekend te kunnen worden en de verhouding tussen horizontale en verticale schaal om te compenseren voor een rechthoekig scherm. Deze waarden zijn over het algemeen constant. `gluLookAt` neemt als argumenten de positie van de camera, de positie waar de camera naartoe gedraaid is en de as die de richting naar boven aangeeft. Deze waarden willen we veranderen aan de hand van toetsenbord input, zodat we goed zicht kunnen krijgen op het 3D model en de resultaten van de algoritmen.

6.3 Voorbeeld

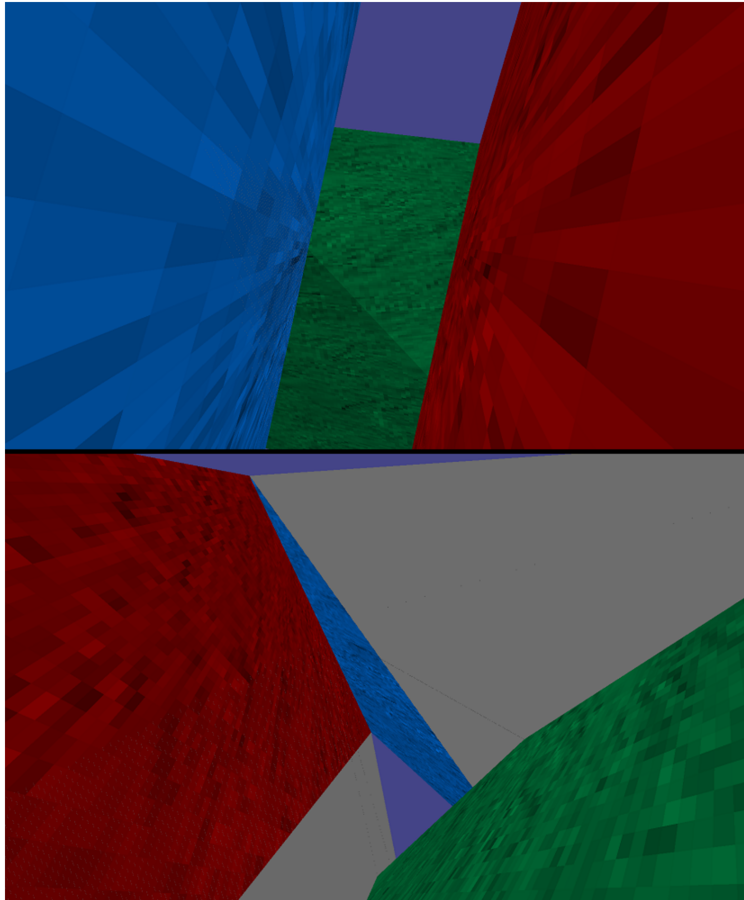
Neem bijvoorbeeld een 3D-model dat bestaat uit twee driehoeken. Dan kunnen we expliciet uitrekenen welke gebieden een camera niet kan zien en visualiseren. Bekijk bijvoorbeeld de onderstaande afbeelding. Als we de camera zoals in het eerste plaatje plaatsen, dan kunnen we het in het grijs gekleurde gebied in het tweede plaatje niet zien.



Met de Simplex module is dit resultaat gemakkelijk te verkrijgen. Hieronder is de code die gebruikt is om het plaatje te genereren (zie ook hoofdstuk 5):

```
import geometry as geom
M = load_model("path")
pol = M.create_polytope()
for shadow in pol:
    for triangle in pol:
        intersection = geom.simplex_S_intersection(triangle + [x, y, z],
                                                   shadow + [x, y, z], 3)
        if len(intersection) >= 4:
            for S in geom.R3_hull_to_polytope(intersection):
                M += S.matrix
```

Hier is `pol` de lijst van alle simplices in het 3D model. Het grijze gebied wordt in de laatste regel aan het model toegevoegd. Een ander voorbeeld is de plaatsing van een camera in het Schönhardt veelvlak:



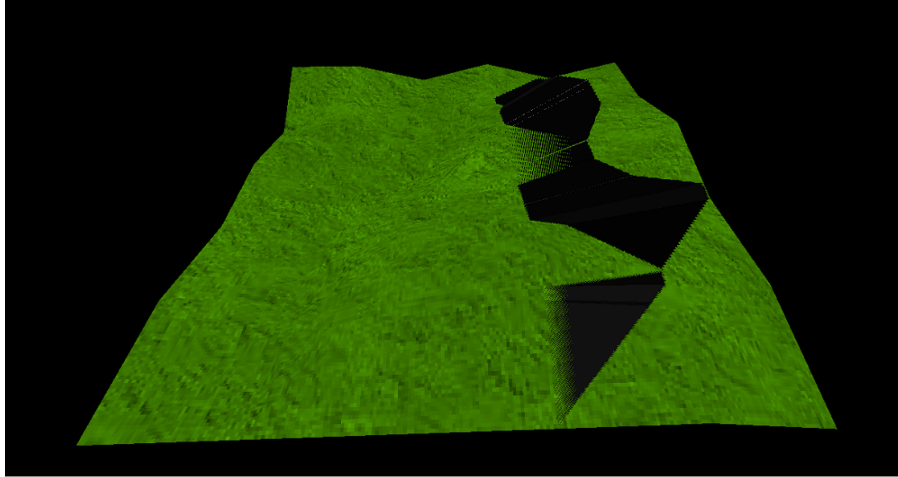
In het bovenste plaatje is de positie van de camera in het Schönhardt veelvlak te zien. In het plaatje is te zien "wat er op de camera te zien is". In het onderste plaatje is met grijs aangegeven welke gebieden voor deze camera onzichtbaar zijn.

6.4 Toepassingen

Om van dit voorbeeld naar een kunstgalerij te gaan is niet moeilijk. Men zou bijvoorbeeld in elk van de grijze gebieden op een hoekpunt een bewaker kunnen plaatsen om de hele kunstgalerij te bewaken. Ook kan men controleren of een bepaalde collectie van bewakers voldoende is om de hele kunstgalerij te bewaken. Dit kan men bijvoorbeeld doen door te kijken naar de doorsnede van alle grijze gebieden voor elke bewaker. Als deze doorsnede leeg is, dan bewaken de bewakers de kunstgalerij. Een doorsnede tussen twee grijze gebieden G_1 en G_2 is gemakkelijk te bepalen. Kijk eerst of de randen van deze gebieden elkaar snijden. Zo niet, controleer of $G_1 \subseteq G_2$ door na te gaan of elk hoekpunt van G_1 in G_2 ligt. Dit kan men doen door voor alle lijnstukken xy met $x \in G_1, \quad y \in G_2$

te checken of deze lijnstukken de rand van G_2 snijden.

Er zijn ook nog andere toepassingen te bedenken. Stel dat men bijvoorbeeld een drone wil gebruiken om een stuk heuvelachtig terrein te bewaken. Dan kan men eenvoudig de gebieden uitrekenen die de drone niet kan zien, zie onderstaande afbeelding



De toepassingen van de module reiken verder dan alleen het kunstgalerijprobleem of varianten daarvan. Men zou bijvoorbeeld voor een natuurkundige simulatie kunnen bepalen waar en wanneer een object in de vorm van een polytoop een ander object in de vorm van een polytoop raakt. Hiervoor hoeven de polytopen niet convex te zijn, iets wat vaak een vereiste is voor dergelijke algoritmen. Ook zou men bijvoorbeeld het pad van een lichtstraal in een ruimte met spiegels kunnen uitrekenen, iets dat handig is voor mooie lichteffecten in games en films.

7 Conclusie

Een wiskundige zal altijd kijken naar mogelijke generalisaties van stellingen die op het eerste gezicht specifiek lijken. De stelling van Chvátal heeft geen mooie generalisatie naar 3 of meer dimensies. Dat feit is niet alleen zeer verrassend, het geeft ook een dieper inzicht in de verschillen tussen computationele meetkunde in 2D en 3D. Het is echter ook zo dat veel zaken hetzelfde werken in 3D en 2D. Hoofdstuk 4 geeft daarin een aantal voorbeelden van berekeningen in de computationele meetkunde die voor elke dimensie werken. Deze algoritmen blijken wel degelijk te kunnen worden gebruikt bij het benaderen van het kunstgalerijprobleem en geven daarbij analyses en (zij het sub-optimale) oplossingen. Door de algoritmen in een computerprogramma te zetten is het gemakkelijk geworden om het kunstgalerijprobleem in $\mathbb{R}^n$ verder te analyseren. Verder leert dit onderzoek niet alleen meer over kunstgalerijen, maar ook over computationele meetkunde in het algemeen. Gezien de grote praktische toepasbaarheid ervan zal de module ook gebruikt kunnen worden in diverse andere gebieden.

Referenties

[1] Carsten Thomassen. *The Jordan-Schönflies Theorem and the Classification of Surfaces*. The American Mathematical Monthly v99 n2 (199202): 116-130, 1992.

[2] Grünbaum, Branko, and Geoffrey C. Shephard. "Convex polytopes." *Bulletin of the London Mathematical Society* 1.3 (1969): 257-300.

[3] O'rourke, Joseph. *Art gallery theorems and algorithms.* Vol. 57. Oxford: Oxford University Press, 1987.

[4] Kahn, Jeff, Maria Klawe, and Daniel Kleitman. "Traditional galleries require fewer watchmen." *SIAM Journal on Algebraic Discrete Methods* 4.2 (1983): 194-206.

[5] Michael, T. S. "Guards, galleries, fortresses, and the octoplex." *The College Mathematics Journal* 42.3 (2011): 191-200.

[6] Brodzik, M. L. "An algorithm for the numerical detection of simplex overlap." *Applied mathematics letters* 8.4 (1995): 13-18.

[7] Rockafellar, R. Tyrrell. *Convex analysis.* No. 28. Princeton university press, 1970.

8 Appendix

Code voor de Simplex-module.

```
import numpy as np
import itertools
from scipy.optimize import linprog

#%%

class Simplex:
    def __init__(self, matrix, color = (255,255,255)):
        self.matrix = np.array(matrix)
        self.space = len(matrix[0])
        self.dimension = np.shape(matrix)[0]*(self.space != 0)
        self.color = color

    def __str__(self):
        string = "("
        for i in range(self.dimension):
            if i != self.dimension - 1:
                string += str(self[i]) + ", "
            else:
                string += str(self[i])
        string += ")"
        return string

    def __getitem__(self, key):
        return self.matrix[key]

    def __setitem__(self, key, value):
        self.matrix[key] = value

    def __add__(self, other):
        return Simplex(np.append(self.matrix.flatten(),
            other).reshape(self.dimension+1,self.space),color =
            self.color)

    def __delitem__(self, key):
        self.matrix = np.delete(self.matrix, key)
        self.dimension -= 1

    def __contains__ (self, item):
        return np.any(np.sum(np.isin(self.matrix,item),axis = 1) ==
            self.space)

    def copy (self):
        return Simplex(np.copy(self.matrix), color = self.color)

    def delete_vertex (self, index):
```

```
    return Simplex(np.delete(self.matrix,index, axis = 0),color =
        self.color)

def remove_duplicates (self, return_index = False):
    if return_index:
        return np.unique(self.matrix,axis = 0,return_index = True)[1]
    return Simplex(np.unique(self.matrix,axis = 0))

def merge (self, other):
    return Simplex (np.unique (np.append (self.vertices,
        other.vertices)), color = self.color)

def get_faces (self):
    return np.delete(np.tile(self.matrix, (self.dimension, 1)),
        np.arange(0, self.dimension**2, self.dimension + 1), axis =
        0).reshape(self.dimension,self.dimension-1,self.space)

def get_line_segments (self):
    return itertools.combinations(self.matrix,2)

def get_parents (self, grandparent):
    faces = grandparent.get_faces()
    return
        faces[np.any(np.all(np.isin(faces,self.matrix),axis=2),axis=1)]

def connected (self, other):
      return len(np.setdiff1d(self.matrix.view([('',
          self.matrix.dtype)] * self.matrix.shape[1]),
                              other.matrix.view([('',
                                  other.matrix.dtype)] *
                                  other.matrix.shape[1]), assume_unique
                                  =
                                  True).view(self.matrix.dtype).reshape(-1,
                                  self.matrix.shape[1])) ==
                                  self.dimension - 1

def affine_independent (self):
    return np.linalg.matrix_rank (self.translated_matrix ()) ==
        self.dimension - 1

def subset_plane(self):
    return not np.any(self[:,0])

def translate(self, vector):
    return Simplex(self.matrix + vector, color = self.color)

def transform(self, mat):
    return Simplex(self.matrix @ mat, color = self.color)

def translated_matrix(self, respect = 0):
```

```
        return self.matrix - self.matrix[respect]

    def direction_vectors(self, respect = 0):
        return np.delete(self.translated_matrix(respect), respect, axis
            = 0)

    def project(self, axis):
        return
            Simplex(self.matrix[:,np.delete(np.arange(self.space),axis)],
            color = self.color)

#%%
def sq_norm(vector):
    return np.inner(vector, vector)

def point_in_simplex(point, simplex):
    V_0 = simplex.matrix[0]
    E_i = simplex.direction_vectors()
    if simplex.space + 1 != simplex.dimension:
        print("System is overdetermined!")
        return False
    D = np.linalg.det(E_i)
    sgnD = np.sign(D)
    tile = np.tile(E_i, (simplex.space, 1))
    tile[np.arange(0, simplex.space*simplex.space, simplex.space + 1)] =
        point - V_0
    dets = np.linalg.det(tile.reshape(simplex.space, simplex.space,
        simplex.space))
    return np.all((sgnD*dets)>=0) and sgnD*np.sum(dets)<=abs(D)

def points_opposite(P, Q, base, directions, return_P_in_plane = False):
    D = np.linalg.det(np.append(directions, P -
        Q).reshape(directions.shape[0]+1, directions.shape[1]))
    sgnD = np.sign(D)
    Pdet = np.linalg.det(np.append(directions, P -
        base).reshape(directions.shape[0]+1, directions.shape[1]))
    theta = sgnD*Pdet
    if return_P_in_plane:
        return theta >= 0 and theta <= abs(D) and D!=0, Pdet == 0
    return theta >= 0 and theta <= abs(D) and D!=0

def simplex_intersection(simplex1, simplex2, err = 0.01):
    X=[]
    for base1 in range(simplex1.dimension):
        for base2 in range(simplex2.dimension):
            U0V0 = simplex2[base2] - simplex1[base1]
            stack = np.append(simplex1.direction_vectors(base1),
                -simplex2.direction_vectors(base2), axis = 0)
            for seq in itertools.product([False,True], repeat =
                simplex1.dimension + simplex2.dimension - 2):
```

```
                sequence = np.array(seq)
                theta_count = np.sum(sequence[:simplex1.dimension - 1])
                matrix = stack[sequence]
                size = len(matrix)
                if size >= simplex1.space:
                    rank = np.linalg.matrix_rank(matrix)
                    if rank == np.linalg.matrix_rank(np.append(matrix,
                         UOVO).reshape(size + 1, simplex1.space)) and rank
                         == size:
                        sol, _, _, _ = np.linalg.lstsq(matrix.transpose(),
                             UOVO, rcond = None)
                        if np.all(sol >= -err) and
                             np.sum(sol[:theta_count]) <= 1 + err and
                             np.sum(sol[theta_count:]) <= 1 + err:
                            point = simplex1[base1] +
                                 (matrix[:theta_count].transpose() @
                                 sol[:theta_count])
                            close_point = False
                            for v in X:
                                if sq_norm(v - point) < err:
                                    close_point = True
                                    break
                            if not close_point:
                                X += [point]
    return X

def simplex_S_intersection(simplex, S, base, err = 0.01):
    X = simplex_intersection(simplex, S.delete_vertex(base))
    for varbase in range(simplex.dimension):
        UOVO = S[base] - simplex[varbase]
        stack = np.append(simplex.direction_vectors(varbase),
             -S.direction_vectors(base), axis = 0)
        for seq in itertools.product([False, True], repeat =
             simplex.dimension + S.dimension - 2):
            sequence = np.array(seq)
            theta_count = np.sum(sequence[:simplex.dimension - 1])
            matrix = stack[sequence]
            size = len(matrix)
            if size >= simplex.space:
                rank = np.linalg.matrix_rank(matrix)
                if rank == np.linalg.matrix_rank(np.append(matrix,
                     UOVO).reshape(size + 1, simplex.space)) and rank ==
                     size:
                    sol, _, _, _= np.linalg.lstsq(matrix.transpose(),
                         UOVO, rcond = None)
                    if np.all(sol >= err) and np.sum(sol[:theta_count]) <=
                         1 + err and np.sum(sol[theta_count:]) >= 1 - err:
                        point = simplex[varbase] +
                             (matrix[:theta_count].transpose() @
                             sol[:theta_count])
```

```
                        close_point = False
                        for v in X:
                            if sq_norm(v - point) < err:
                                close_point = True
                                break
                        if not close_point:
                            X += [point]
    return X

def simplex_intersection_bool(simplex1, simplex2, err = 0.01):
    A = np.append(np.hstack((simplex1.matrix,
        np.ones((simplex1.dimension, 1)))),
                np.hstack((-simplex2.matrix,
                    -np.ones((simplex2.dimension, 1)))), axis =
                    0).transpose()
    res = linprog(-np.ones(A.shape[1]), A_eq = A, b_eq =
        np.zeros(A.shape[0]), bounds = (0, 1))
    return res.success and sq_norm(res.x) >= err

def projection_on_plane(point, base, directions):
    return base + np.dot(np.linalg.solve(directions @
        directions.transpose(), np.dot(directions, point - base)),
        directions)

def line_plane_intersection(P, Q, base, directions, err = 0):
    PQdet = np.linalg.det(np.append(directions, P -
        Q).reshape(directions.shape[0]+1, directions.shape[1]))
    PBdet = np.linalg.det(np.append(directions, P -
        base).reshape(directions.shape[0]+1, directions.shape[1]))
    if PQdet == 0:
        return None
    theta = PBdet/PQdet
    if theta >= -err and theta <= 1 + err:
        return P + theta*(Q - P)
    return None

def line_plane_barycentric(P, Q, base, directions):
    try:
        bar = np.linalg.solve(np.append(directions, P -
            Q).reshape(directions.shape[0]+1,
            directions.shape[1]).transpose(), P - base)
    except np.linalg.LinAlgError:
        return None
    return bar

#%%

def R3_hull_to_polytope(hull):
    simplexlist = []
    indices = np.arange(len(hull))
```

```
for vertextriple in itertools.combinations(indices, 3):
    is_boundary = True
    base = hull[vertextriple[0]]
    dir1 = hull[vertextriple[1]]
    dir2 = hull[vertextriple[2]]
    diff1 = dir1 - base
    diff2 = dir2 - base
    for pair in itertools.combinations(np.setdiff1d(indices,
         vertextriple), 2):
        if not line_plane_intersection(hull[pair[0]], hull[pair[1]],
             base, np.array([diff1, diff2]), err = -0.01) is None:
            is_boundary = False
    if is_boundary:
        simplexlist += [Simplex(np.array([base, dir1, dir2]))]
return simplexlist
```