Academiejaar 2010–2011
Schoonmeersstraat 52 - 9000 Gent

Departement Toegepaste Ingenieurswetenschappen

Ray-acoustic modeling, based on Adaptive Beam Tracing

Masterproef voorgedragen tot het behalen van het diploma van Master in de industriële wetenschappen: informatica

Christophe VAN AUSELOOS

Interne promotor: *Koen VAN DE WIELE*
Externe promotoren: *Gert MASSA en Manu DE GEEST*

1. Woord vooraf

Deze masterproef vormt het sluitstuk van mijn masteropleiding in de Toegepaste Industriële Wetenschappen Informatica aan het departement Industriële Wetenschappen van de Hogeschool Gent. Het onderwerp van deze masterproef is "Ray-acoustic modeling, based on Adaptive Beam Tracing".

Deze paper is het verslag van mijn stage bij LMS International. Ik ontwierp er software om akoestiek te simuleren. De ontworpen software zal in de toekomst gebruikt worden in één van hun commerciële toepassingen (Raynoise, Virtual.Lab).

Ik heb voor dit onderwerp gekozen omdat het mij een perfecte combinatie leek tussen de ingenieurs wetenschappen (fysica) en de informatica (programmering). Beide onderwerpen zijn grondig aanbod gekomen tijdens mijn opleiding.

Ik heb mijn masterproef tot een goed einde gebracht dankzij de hulp van verschillende mensen.
In de eerste plaats wil ik mijn externe promotoren, Gert Massa en Manu De Geest, bedanken voor de ondersteuning en de zéér leerrijke ervaring. Ook wil ik mijn interne promotor, Koen Van De Wiele, bedanken voor de ondersteuning en het opvolgen van dit project.

Tenslotte wil ik graag mijn familie, mijn vriendin en mijn vrienden bedanken die mij altijd onvoorwaardelijk zijn blijven steunen en mij ook op moeilijke momenten het nodige vertrouwen hebben gegeven om door te gaan. Zonder hun steun had ik zeker niet hetzelfde eindresultaat kunnen bekomen. In het bijzonder wil ik graag Jo Van Biesbroeck, Nadine de Beauffort, Guido Van Auseloos en Marina Denruyter bedanken voor de tijd die ze vrijgemaakt hebben om deze scriptie grondig na te lezen.

Christophe Van Auseloos
Kessel-lo, 2010-2011

2. Abstract

Ray-Acoustic modeling, based on Adaptive Beam Tracing

Two techniques exist for simulating acoustics, they are known as Geometrical Acoustic Method (GA) and Numerical Computational Method (NCM).

GA approaches are derived using asymptotic approximations of the wave equation in the infinite frequency limit. The basic assumption is that sound propagates in linear paths, similar to light.

In the nineties, LMS International developed a hybrid GA approach referred to as Triangular Beam Tracing Method (TBT), which assumes continuous bundles of sound rays. This software is integrated in the LMS Virtual.Lab product suite (based on Dassault Systemes' CATIA V5 environment).

The limitations of the current TBT software are:
1) The lack of accuracy when emitting large bundles of sound rays.
2) The inability to handle edge diffraction in higher reflection/diffraction orders.
3) The poor performance, especially on multi-core CPUs.

The aim of this project is to design and implement a new GA solver to improve or eliminate these limitations.

Ray-Akoestische modellering, gebaseerd op Adaptive Beam Tracing

Er bestaan twee technieken voor het simuleren van akoestiek, de Geometrical Acoustic Method (GA) en de Numerical Computational Method (NCM).

GA benaderingen zijn afgeleid met behulp van asymptotische benaderingen van de golf vergelijking in een oneindige frequentie limiet. De basisonderstelling stelt dat geluid zich voortplant in lineaire paden, zoals licht.

LMS International ontwikkelde in de jaren negentig een hybride GA benadering, de Triangular Beam Tracing Method (TBT), die gebaseerd is op continue bundels van geluidsstralen. Deze software is nog steeds geïntegreerd in het LMS Virtual.Lab product (gebasseerd op Dassault Systemes' CATIA V5 omgeving).

De beperkingen van de huidige TBT zijn als volgt:
1) De onnauwkeurigheid bij het emitteren van grote bundels geluidsstralen.
2) De onmogelijkheid om hoek diffractie uit te voeren in hogere reflectie of diffractie orders.
3) De zwakke performantie, vooral op multi-core CPU's

Het doel van dit project is het ontwerpen en implementeren van een nieuwe GA-solver die deze beperkingen verbetert of elimineert.

3. Inhoudsopgave

4. Figuren

5. Codevoorbeelden

6. Het bedrijf: LMS International en haar producten

LMS International is een spin-off bedrijf van de KULeuven, opgestart in 1980. Het bedrijf is gespecialiseerd in onderzoek naar akoestiek, vibraties, kwaliteit, durabiliteit en veiligheid van systemen. Oorspronkelijk was het een klein bedrijf, gevestigd op het industrieterrein in Haasrode (Leuven). Ondertussen heeft LMS International 900 werknemers in 31 kantoren en meer dan 5000 klanten verspreid over de hele wereld. Zo is LMS bijvoorbeeld actief in België, Nederland, Frankrijk, India en Amerika. De hoofdzetel is nog steeds in Haasrode, waar momenteel 400 mensen tewerkgesteld worden. Door de brede waaier aan specialiteiten van het bedrijf is LMS International in vele markten thuis, namelijk in de automobiel -, bouw -, ruimtevaart - en luchtvaartsector.

LMS International heeft 3 basis softwarepakketten:

a. LMS Test.Lab

Test.Lab wordt gebruikt om de structuur, vibraties en het akoestische design van systemen te testen. Dit gebeurt door middel van meetapparatuur, eveneens door LMS International ontwikkeld.
Een voorbeeld hiervan is het analyseren van het geluid van een 4-takt motor waarbij elk onderdeel afzonderlijk geanalyseerd wordt.

b. LMS Imagine.Lab (AMESim)

Imagine.Lab is een 1D-simulatie softwarepakket waarmee systemen geanalyseerd worden op hun elektrisch, hydraulisch, pneumatisch en mechanisch gedrag.
Dit pakket wordt voornamelijk gebruikt in de automobielsector.
Hieronder volgen enkele onderdelen:

- Imagine.Lab Powertrain Transmission
- Imagine.Lab Internal Combustion Engine
- Imagine.Lab Vehicle System Dynamics

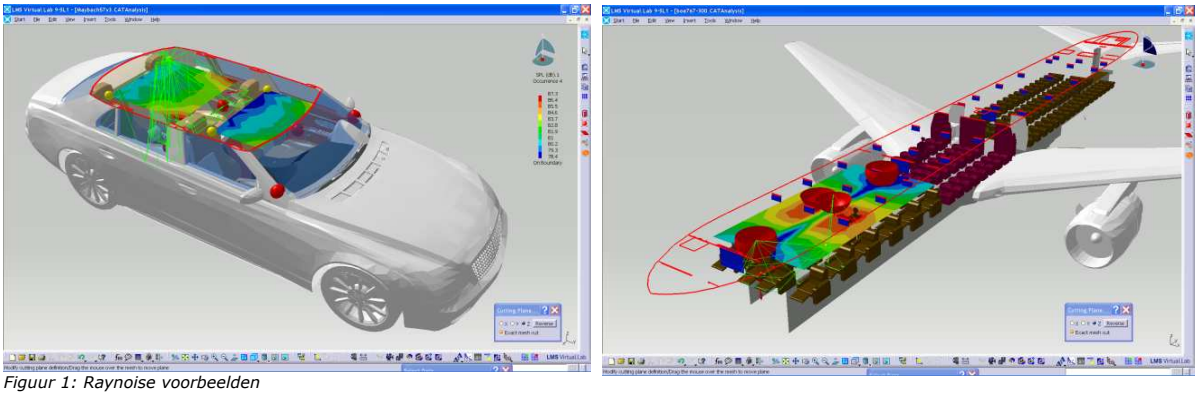
c. LMS Virtual.Lab

Virtual.Lab is een functioneel performantie simulatie softwarepakket waarmee 3D-systemen geanalyseerd worden op hun akoestische- en structurele eigenschappen net zoals op de eigenschappen van de vibratie en de duurzaamheid ervan. Dit programma heeft voor elke systeemeigenschap een specifiek onderdeel. Zo zijn Raynoise en Sysnoise de onderdelen waarin akoestiek behandeld wordt. Het onderdeel voor structuur is Structures en dat voor duurzaamheid is Durability.
Virtual.Lab is gebaseerd op de CATIA V5 omgeving, een virtuele omgeving waarin men drie dimensionale objecten kan ontwerpen ("Computer Aided Three dimensional Interactive Application").

Deze masterproef heeft betrekking op het analyseren van de systeemeigenschap akoestiek, meer bepaald het onderdeel Raynoise van Virtual.Lab.

i. Raynoise

Akoestiek is een belangrijk aspect bij de constructie van operagebouwen, vliegtuigen, auto's, ruimteschepen, … . Het fysisch testen van akoestiek gaat natuurlijk enkel wanneer een project voltooid is. Sommige constructies zijn zo tijdrovend en/of duur dat ontwikkelaars op voorhand moeten weten of de akoestiek van hun constructie optimaal is. Om dit te kunnen nagaan heeft LMS International in de jaren '90 Raynoise ontwikkeld. Raynoise is een deel van Virtual.Lab dat de akoestische eigenschap van een virtueel object berekent. Het visuele aspect van het simulatie pakket behoort tot CATIA V5 en is geprogrammeerd in C. De solver, het aspect die de akoestiek berekent is geprogrammeerd in Fortran.



Figuur 1: Raynoise voorbeelden

Momenteel gebruikt de solver een Triangular Beam Tracing techniek. Hierbij wordt elke geluidsgolf als een driehoekig object voorgesteld. Elk verzonden geluidsobject wordt groter naarmate de afstand tussen de bron en het object vordert. Indien een geluidsobject met een oppervlak botst, wordt het gereflecteerd en verliest het eventueel energie door absorptie.

Het doel van deze masterproef was een nieuwe techniek te ontwerpen en te implementeren die meer nauwkeurige resultaten berekent en een betere performantie heeft.

7. Het onderwerp: simulatie van akoestiek

Het doel van de virtualisatie van akoestiek is het analyseren van de intensiteit van een geluidsbron ten opzichte van een bepaald object.
Een voorbeeld hiervan zijn de luidsprekers van een auto en het menselijk oor. Hierbij kan men testen hoe het materiaal van het interieur van een auto de verzonden geluidsgolf wijzigt door bijvoorbeeld absorptie, of hoeveel echo er ontstaat door reflectie van geluidsgolven.
Een perfecte geluidsstraal heeft minimale absorptie en echo.

Virtualisatie van akoestiek kan onderverdeeld worden in twee groepen:

a. Geometrical Acoustics (GA)

Geometrical Acoustics stelt de bewegingen van een geluidsgolf virtueel voor.
Een geluidsgolf wordt voorgesteld als een geluidsstraal met begin en eindpunt.
Een geluidsbron wordt voorgesteld als een sferisch object dat in bepaalde richtingen geluidsstralen van eenzelfde intensiteit verzendt. De verzonden geluidsstralen worden dan virtueel gereflecteerd, geabsorbeerd of afgebogen (diffractie).
De geluidseigenschappen moeten specifiek worden gedefinieerd in de virtualisatie code.

Om de intensiteit van een golf ten opzichte van een object te bepalen worden er virtuele microfonen (fieldpoints) geplaatst. Het doel van Geometrical Acoustics is de verschillende wegen van de geluidsbron tot een fieldpoint te zoeken. Later worden deze wegen gebruikt om de intensiteit van de geluidsgolf op de microfoon te bepalen.

Het voordeel van deze techniek is dat er frequentie onafhankelijk met de golven gewerkt wordt. Er wordt ook relatief weinig geheugen gebruikt, want enkel de wegen van de geluidsbron naar een fieldpoint zijn belangrijk.
Het nadeel is dat deze techniek minder accuraat is dan zijn tegenhanger: de Numerical Computational Method.

De laatste jaren zijn er heel wat studies gemaakt op het vlak van simulatie van akoestiek binnen de Geometrical Acoustics wereld. Er bestaan veel Geometrical Acoustics algoritmes; Ray Tracing, Beam Tracing, Adaptive Beam Tracing, Frustum Tracing, ...

b. Numerical Computational Method (NCM)

De Numerical Computational Method berekent de akoestische intensiteit van een volledige virtueel object. Dit gebeurt door middel van een mesh. Een mesh is een opdeling van een virtueel object per minimale oppervlakte eenheid. De mesh wordt bewaard in een matrix.
Numerical Computational Method analyseert deze matrix met behulp van een formule (integraal bewerkingen) die de akoestische eigenschappen ten opzichte van het virtueel object berekent.
De toegepaste formule is eigen aan de gebruikte techniek (BEM, FEM, ...).

Het voordeel van deze techniek is dat de oplossing steeds zeer accuraat is. Ook het feit dat er geen extra berekeningen moeten gebeuren in verband met diffractie, absorptie en reflectie is een voordeel. Alle geluidseigenschappen worden in rekening gebracht in de formule.

Het nadeel is dat voor een groot object de mesh zeer groot kan zijn, wat als gevolg heeft dat de matrix veel geheugen nodig heeft.

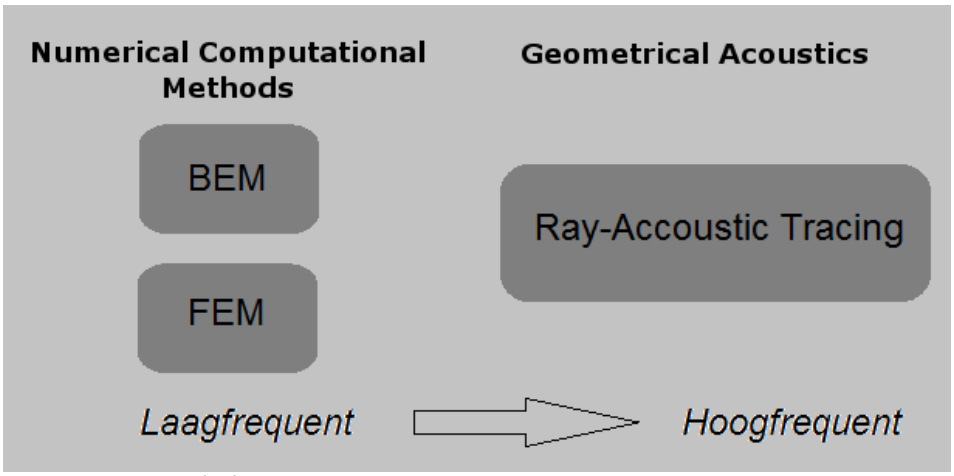
Een ander nadeel is dat de mesh afhankelijk is van de geluid frequentie. Om het juiste resultaat uit een mesh te verkrijgen moet de minimale oppervlakte van een object klein genoeg zijn om $1/6^{de}$ van een geluidsgolf te kunnen analyseren. Dus voor een geluidsstraal met een hoge frequentie moet er een mesh worden gebruikt met veel kleine opdelingen, wat dus veel geheugen vraagt.

De bekendste NCM's zijn Boundary Element Method (BEM) en Finite Element Method (FEM). Omdat deze thesis gericht is op Geometrical Acoustics worden deze technieken niet verder uitgelegd.

c. Vergelijking

Numerical Computational Methods zijn steeds een zeer goede representatie van een virtuele oplossing. Maar vanwege het enorm geheugengebruik is het niet mogelijk om enkel NCM's te gebruiken voor virtuele akoestische problemen. NCM's zullen voornamelijk gebruikt worden op kleinere modellen of met laagfrequente golven.

Ten opzichte van NCM's hebben Geometrical Acoustic Methods gemiddeld minder geheugen nodig, maar zijn dan ook weer minder accuraat. GA's worden daarom dan ook gebruikt om hoogfrequente golven te virtualiseren, toch is het ook perfect mogelijk om met laagfrequente golven te werken omdat GA frequentie onafhankelijk is.



Figuur 2: Toepassingsgebieden GA en NCM

8. De opdracht: simulatie van akoestiek, gebaseerd op Adaptive Beam Tracing

Sinds de ontwikkeling van de huidige GA-solver in Raynoise (1998) zijn er redelijk wat studies gebeurd naar nieuwe GA-solvers. Daarom besliste LMS om de solver van Raynoise te vervangen door een nieuwe die nauwkeuriger te werk gaat en een betere performantie heeft.

De solver wordt stapsgewijs vervangen. De eerste stap is een nieuwe GA-solver ontwerpen en implementeren. De volgende stap is het omzetten van een geluidspad naar intensiteitwaarden, dit gebeurt door middel van voorgedefinieerde formules. De laatste stap is dan de Adaptive Beam Tracing solver te integreren binnen de Raynoise werkomgeving.

Mijn opdracht bestaat uit de eerste stap, het zoeken van geluidspaden van bron naar microfoon. De GA-solver moet gebaseerd zijn op de Adaptive Beam Tracing methode (ABT). Naast de ABT methode moeten ook de belangrijkste golf eigenschappen, zoals reflectie en diffractie, worden gesimuleerd.
De eerste stap bestaat echter enkel uit het zoeken van geluidspaden. Er wordt dus nog geen rekening gehouden met de intensiteit van een geluidsgolf. Mijn opdracht zal dus ook geen rekening houden met de absorptie eigenschappen van het materiaal waarop een geluidsgolf botst. Alle geluidsgolven die nu worden berekend hebben een maximale intensiteit en hebben ook bijvoorbeeld een perfecte reflectie zonder enige absorptie van het materiaal.

9. Enkele basisbegrippen van akoestiek

Voor de bespreking van mijn opdracht is het nodig om enkele basiseigenschappen en –begrippen van akoestiek te definiëren of te verduidelijken.

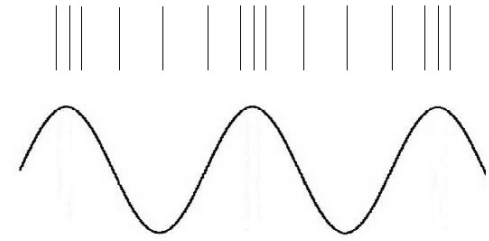
Geluid is een fenomeen waarmee de mens dagelijks geconfronteerd wordt: de wekker, communiceren met collega's, de radio, de televisie, ... Een wereld zonder geluid is dan ook ondenkbaar. Bij veel communicatievormen is geluid immers van essentieel belang. Om deze reden is het dus belangrijk dat het geluid goed overkomt. Bij LMS International wordt er continu gezocht naar manieren en technieken om geluid zo goed mogelijk over te brengen.

Akoestiek is een wetenschap die bestaat uit de studie van trillingen in verschillende omgevingen zoals water, lucht, steen. Een trilling bestaat uit het samendrukken en expanderen van moleculen.

De belangrijkste eigenschappen van geluidsgolven, binnen de grenzen van het onderwerp van deze thesis, zijn: de sterkte van het geluid (dB), de frequentie (f), de reflectie en de diffractie ervan.

a. De geluidsgolf en haar kenmerken

Een geluidsgolf plant zich voort als een oscillatie door moleculen samen te drukken.



Deze oscillatie kan voorgesteld worden met behulp van een sinusoïdale golf, waarbij de bovenste pieken het samendrukken en de onderste pieken het expanderen van moleculen voorstellen (*Figuur 3: De geluidsgolf*).

Figuur 3: De geluidsgolf

Door deze oscillatie mag een geluidsgolf niet verward worden met een elektromagnetische golf. Bijvoorbeeld een radiogolf of een lichtgolf. Een elektromagnetische golf is een zuivere sinusoïdale golf. Een geluidsgolf wordt voorgesteld als een sinusoïdale golf, maar is eigenlijk een oscillatie van samendrukkende moleculen.

Omdat geluid bewegende moleculen nodig heeft om zicht voort te bewegen, heeft het dus een medium nodig. Dit kan lucht of water zijn, maar ook een harder materiaal zoals steen. Opmerkelijk is dat in de ruimte, waar geen medium is, geen geluid bestaat.

i. Intensiteit

De intensiteit (I) van een golf bepaalt het vermogen dat door een eenheidsoppervlak, loodrecht op de richting van de energiestroom, wordt overgedragen. Intensiteit wordt uitgedrukt in W/m².

In de veronderstelling dat de geluidsbron zijn geluidsstralen als een bol uitzendt, wordt de intensiteit met de volgende formule berekend:

$$I_r = \frac{P_{ac}}{A} = \frac{P_{ac}}{4\pi r^2}$$

Figuur 4: Formule intensiteit

Ir [W/m²]: intensiteit
Pac [W]: de sterkte waarmee de geluidsbron geluid verzendt
A [m²]: oppervlakte

Uit de formule is af te leiden dat de intensiteit in functie staat van de afstand van de geluidsbron tot de ontvanger (1/r²). Als de afstand tot de ontvanger bijvoorbeeld verdubbelt, zal de intensiteit tot een kwart reduceren.

De intensiteit van een geluidsstraal verzwakt in functie van het type materiaal waarop een geluidsstraal botst; of ook door het effect van bijvoorbeeld diffractie op een geluidsstraal.
Dit laatste begrip wordt later besproken.

ii. Decibel

De sterkte van een geluidsgolf wordt bepaald door de intensiteit van de gemeten geluidsgolf te vergelijken met de minimaal hoorbare intensiteit van een geluidsgolf. Volgende formule wordt hiervoor gebruikt:

$$L_{\mathrm{I}} = 10 \log_{10}\left(\frac{I_1}{I_0}\right)$$

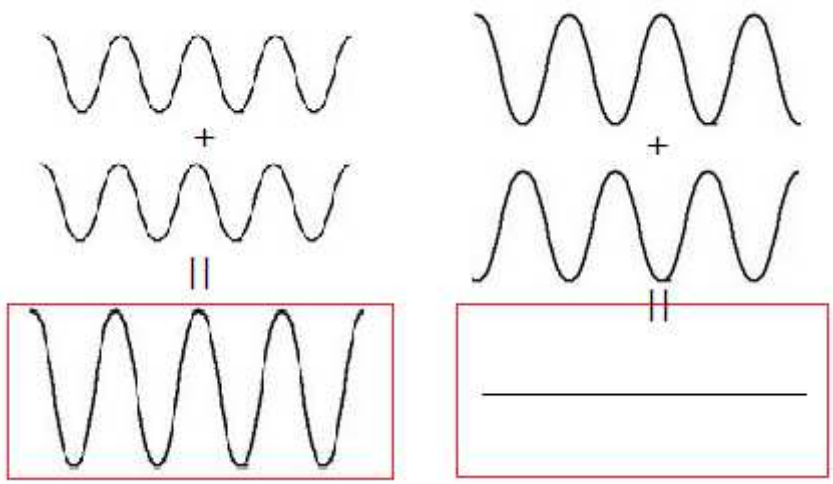
Figuur 5: Formule decibel

L1 [dB]: de geluidssterkte, uitgedrukt in decibel
I1 [W/m²]: de gemeten intensiteit
I0 [W/m²]: is gewoonlijk gelijk aan de minimale intensiteit dat een goed oor kan opvangen
= $1.0 * 10^{-12}$ W/m²

iii. Frequentie

Het aantal trillingen (oscillaties) binnen een bepaalde tijdseenheid wordt uitgedrukt met de term frequentie (f). De tijd nodig voor een volledige oscillatie van een golf noemt men de periode (T).
Een golf die binnen de tijdseenheid veel periodes heeft is hoogfrequent en een golf met weinig periodes binnen de tijdseenheid is laagfrequent.

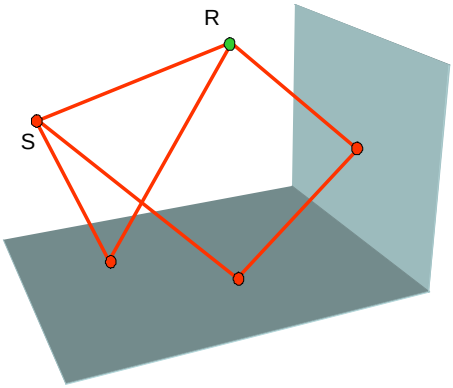
Een belangrijke eigenschap van golven, in verband met frequentie, is dat de frequentie van overlopende golven worden opgeteld. Op deze manier kan men een geluidssignaal versterken of volledig elimineren:



Figuur 6: Optellen golf frequentie

De frequentie die een andere golf volledig elimineert wordt de "tegenfrequentie" genoemd.

iv. Reflectie



Figuur 7: Reflectie

Een geluidsgolf die botst met een object, wordt gereflecteerd. Een object kan uit eender welk materiaal bestaan. De sterkte van de gereflecteerde geluidsgolf vermindert door absorptie van het element. Het materiaal waaruit het element bestaat, speelt hier een belangrijke rol. Een zacht materiaal zal de sterkte van een geluidsgolf meer verzwakken dan een hard materiaal.

Het verschijnsel 'echo' ontstaat door reflectie. Wanneer een teruggekaatste geluidsstraal langer onderweg is dan een directe geluidsstraal, verzonden uit dezelfde geluidsbron, ontstaat er een verschil van ontvangst tijd. Het verschil tussen de tijd van een directe geluidsstraal en een gereflecteerde geluidsstraal vormt de echo.

Zo heeft het interieur van een grote auto meer echo dan van een kleine auto. Omdat een geluidsstraal in een grote auto langer onderweg is voor deze gereflecteerd wordt opgevangen door dezelfde microfoon.

Een ander belangrijk gevolg van reflectie is een gereflecteerde golf die 180° wordt geroteerd. Hierdoor wordt een maxima een minima en een minima een maxima.

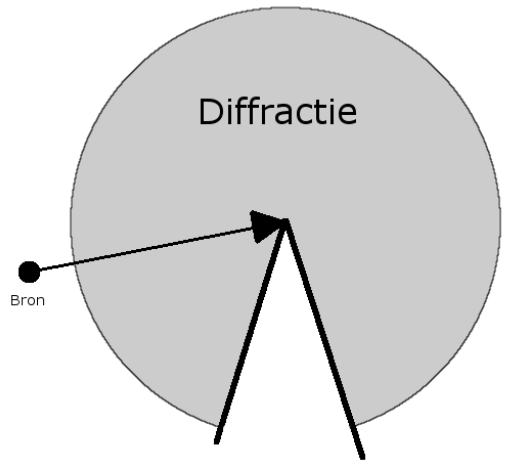
Figuur 8: Reflectie, rotatie

Reflectie wordt gebruikt in verschillende industrieën. Zo wordt de diepte van de zee bijvoorbeeld bepaald met behulp van een echolood. Uit dit echolood wordt een geluidsgolf richting de zeebodem verstuurd. Aan de hand van de tijd die de verzonden geluidsgolf nodig heeft om te reflecteren op de zeebodem en opnieuw tot het echolood te geraken, wordt er berekend hoe diep de zeebodem is.

v. Diffractie, gebaseerd op de Geometrical Theory of Diffraction

De Geometrical Theory of Diffraction definieert dat een gedeelte van de scheidingslijn een nieuwe bron zal vormen. De bron zal geluidsstralen sturen in alle mogelijke richtingen, begrensd door het intersectie- en het buurobject.

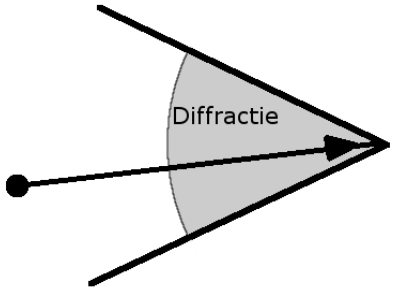
Figuur 9 toont een invallende, vrije golf (zie later: *1) Creeping waves*) die botst op de scheidingslijn van twee objecten. De twee objecten staan onder een bepaalde hoek, het object dat het dichtst bij de bron staat is het **intersectieobject** en het andere element is het **buurobject**.



Figuur 9: Diffractie

De *Geometrical Theory of Diffraction (GTD)* definieert dat een gedeelte van de scheidingslijn een nieuwe bron zal vormen. De bron zal geluidsstralen sturen in alle mogelijke richtingen, begrensd door het intersectie- en het buurobject.

Merk op dat diffractie ook zorgt voor nieuwe golven die richting de originele bron verstuurd worden, dit effect is meestal vergeten maar is toch van essentieel belang voor een correcte virtualisatie. Dit is bijvoorbeeld van belang bij twee objecten die onder een hoek van minder dan 180° staan ten opzichte van de bron. De opstelling van zo'n objecten heeft de naam "**backward wedge**". Het diffractie effect heeft daarom dan ook als term "**backward diffraction**".

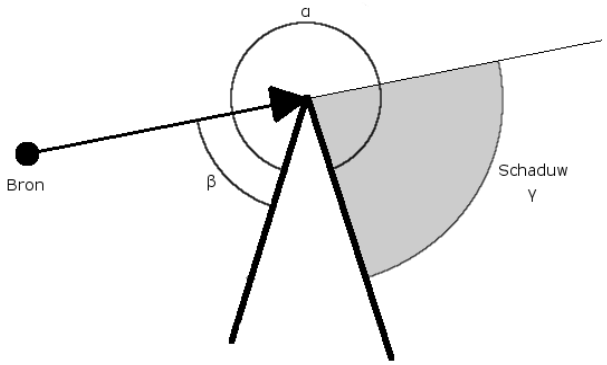


Figuur 10: Backward diffractie

1) Creeping waves, een specifieke diffractie

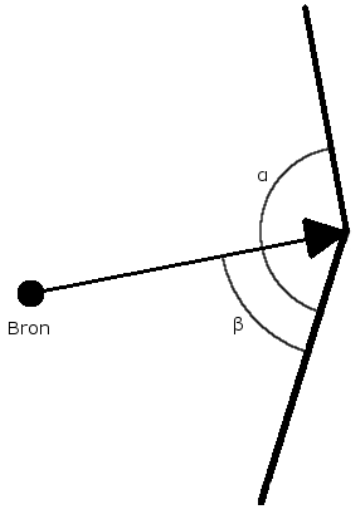
Diffractie creëert dus nieuwe golven in alle mogelijke richtingen tussen het intersectie- en buurobject. Één van deze nieuwe golven is een speciaal geval: de creeping wave (kruipende golf). De creeping wave plant zich voort op het buurobject in de schaduwzone van de invallende golf. Indien het buurobject niet in de schaduwzone ligt, bestaat er ook geen creeping wave.

De schaduwzone (γ) is gedefinieerd door de hoek tussen het intersectie- en het buurobject (α) en de hoek tussen de invallende golf en het intersectieobject (β).
δ = α - β (de hoek tussen de invallende golf en het buurobject)
γ = δ - Π



Indien γ groter of gelijk is aan nul radiaal bestaat er een schaduwzone.

Figuur 11: Creeping wave met schaduwzone



Indien γ kleiner is dan nul bestaat er **geen** schaduwzone en dus ook geen creeping wave.

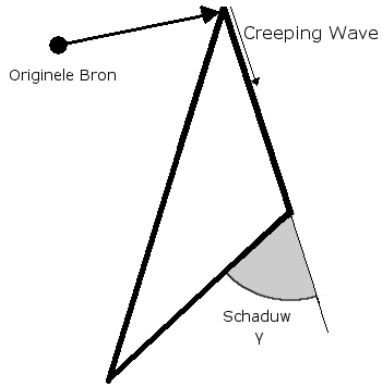
Figuur 12: Creeping wave zonder schaduwzone

2) Diffractie van een creeping wave

Creeping waves kunnen opnieuw worden gediffracteerd.

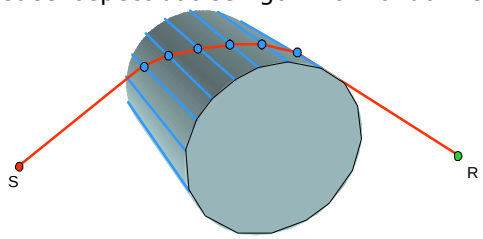
Diffractie van een vrije golf (een niet creeping wave) maakt een nieuwe bron op een gedeelte van de scheidingslijn, begrensd door het intersectie- en buurobject (zie *Figuur 9: Diffractie*).
Diffractie van een creeping wave maakt ook een nieuwe bron op de scheidingslijn waarop de creeping wave botst. De begrenzing van de nieuwe bron is afhankelijk van het wel of niet bestaan van een schaduwzone.

Indien er een **schaduwzone** bestaat wordt de nieuwe bron begrensd door de schaduwzone van de invallende golf en het buurobject, opnieuw ontstaat er een creeping wave in de schaduwzone op het buurobject.
Indien er **geen schaduwzone** bestaat wordt de nieuwe bron begrensd door het intersectie- en buurobject. Omdat er geen schaduwzone bestaat, is hier ook geen creeping wave op het buurobject.



Figuur 13: Diffractie van een creeping wave

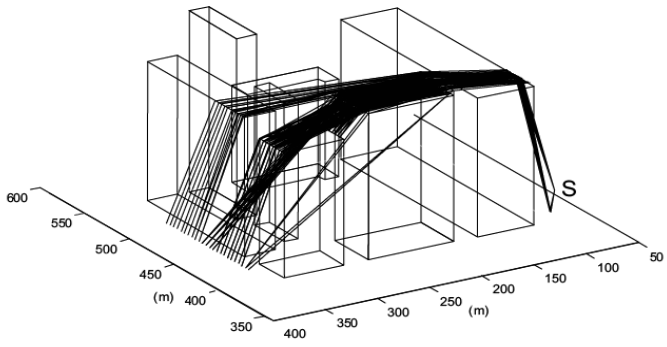
Veel achtereenvolgende creeping waves, die steeds in een schaduwzone ontstaan, zorgen dan voor het visueel aspect dat een golf zich rondom een object buigt.



Figuur 14: Diffractie, creaping waves

Figuur 14 toont een object bestaande uit verschillende rechthoekige objecten. De rechthoekige objecten vormen een bijna cilindervormig oppervlak. Aan weerszijde van het object staan een geluidsbron en –ontvanger. Van de geluidsbron zal een geluidsstraal botsen met de grens tussen twee rechthoekige objecten, dit punt noemt het **attachment point**. De invallende golf zal opsplitsen in verschillende nieuwe golven. Van deze nieuwe golven zullen de creeping waves op hun beurt worden gediffracteerd en nieuwe golven creëren. Deze herhaling blijft oneindig doorgaan (of tot de intensiteit van de geluidsgolf ongeveer gelijk is aan nul). Het punt waar een creeping wave een nieuwe golf genereert die de cilinder verlaat en botst tegen de microfoon noemt het **detachement point**. Door de herhalingen van opsplitsingen zal de ontvanger geluid ontvangen van de geluidsbron via geluidsgolven over (of onder) het cilindervormige oppervlak.

Zo is het mogelijk dat in een vol gebouwde stad geluid van een straat hoorbaar is tot enkele straten verder. Deze geluidsstralen planten zich dan voort over de daken van - en langs de gebouwen.

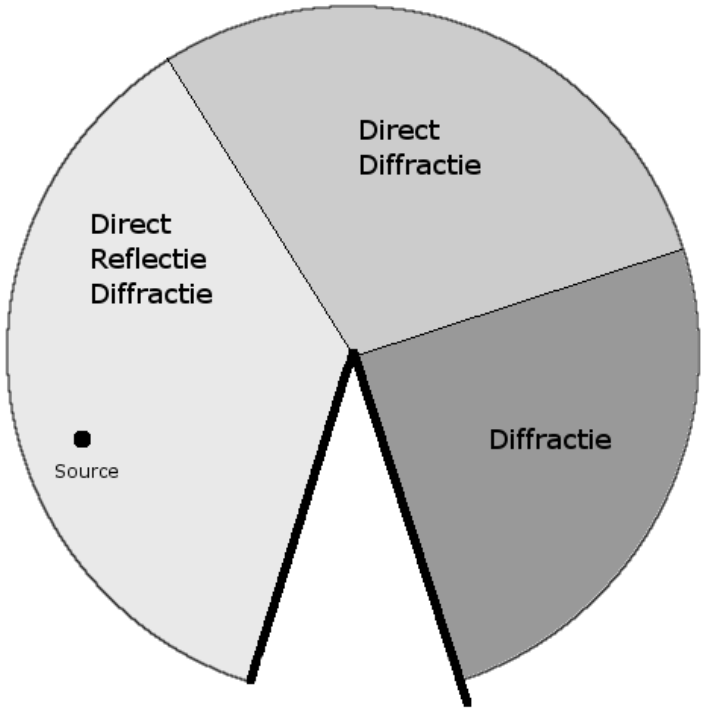


Figuur 15: Diffractie, stad voorbeeld

Om onderscheid te maken tussen de normaal invallende golf en de kruipende golf worden er twee extra termen gedefinieerd. De normale golf krijgt de term "**vrije golf**" en de kruipende golf behoudt de term "**creeping wave**".

vi. Besluit

Een geluidsgolf is onderhevig aan drie basis akoestische eigenschappen:
- De vrije golf die zich voortbeweegt in een medium
- De reflectie van een geluidsgolf op een object
- De diffractie van een geluidsgolf rond de grenslijn van een element



Figuur 16: Samenvatting basisbegrippen

Figuur 16 illustreert een geluidsbron die naast een forward wedge staat. In functie van welke richting de geluidsbron een geluidsstraal uitzendt en waar de microfoon staat, kunnen er drie zones worden gedefinieerd.
Bijvoorbeeld een microfoon in de eerste zone kan bereikt worden door de drie akoestische eigenschappen:
- Direct van de geluidsbron naar de microfoon
- Door reflectie op het intersectieobject
- Door diffractie op de grenslijn

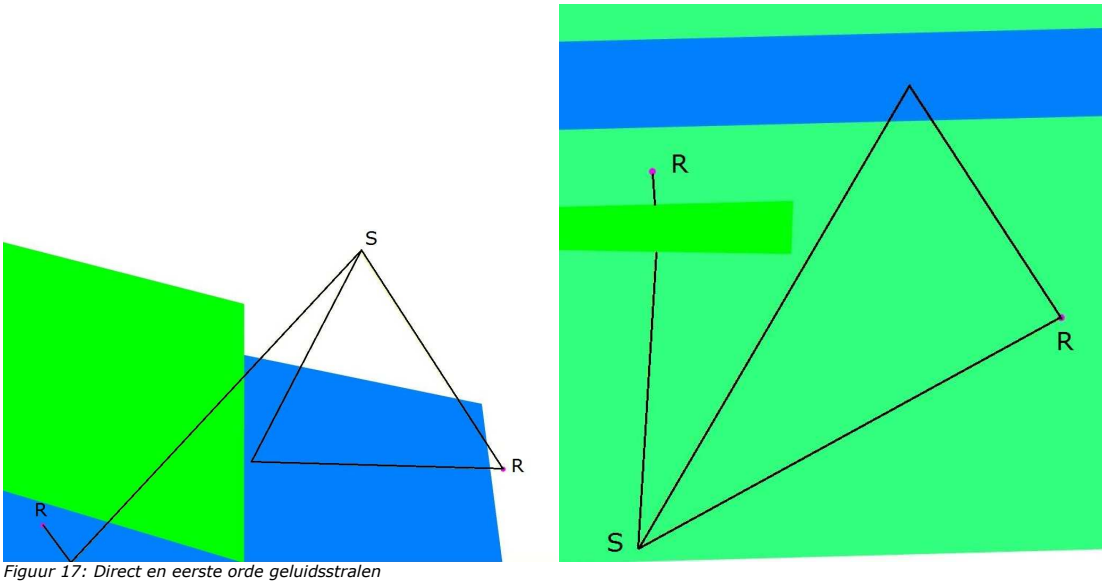
Naast de drie basiseigenschappen bestaat er ook nog de vibroacoustic eigenschap. Deze definieert dat een geluidsgolf een object zodanig kan laten trillen dat de vibratie van het object voor een nieuwe geluidsbron zorgt. Deze eigenschap wordt verder niet besproken in mijn opdracht.

10. Twee Geometrical Acoustics Algoritmes

a. Basisbegrippen van GA's

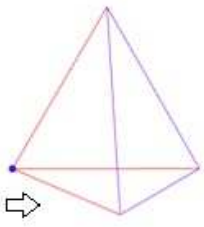
Geometrical Acoustics algoritmes onderzoeken de intensiteit van een verzonden geluidsstraal ten opzichte van een microfoon. Dit gebeurt door virtuele paden te zoeken van de geluidsbron naar de microfoon.

De intensiteit van een geluidsbron, ten opzichte van een microfoon, wordt bepaald door elk pad te onderzoeken op weerkaatsingobjecten en de tijd die nodig was om de microfoon te bereiken.
Een weerkaatsingobject is bijvoorbeeld een muur waarop een geluidsstraal botst en opnieuw wordt verzonden (reflectie). Dit weerkaatsingobject is belangrijk, want dit object absorbeert steeds een bepaalde fractie van een botsende geluidsstraal. De verzendtijd is dan weer nodig omdat de intensiteit van een geluidsstraal kwadratisch vermindert in functie van de afgelegde weg (de afgelegde weg is afleidbaar uit de verzendtijd).



Figuur 17: Direct en eerste orde geluidsstralen

Geometrical Acoustics algoritmes gebruiken **beams** om virtuele microfonen te zoeken. Beams zijn driedimensionaal geometrische vormen en worden gedefinieerd door rays. Een **ray** is een straal die wordt verzonden uit een geluidsbron. De ray is gedefinieerd als een vector met een bron, richting en eventueel botsingspunt. Een beam is dus gedefinieerd door zijn dragende rays.



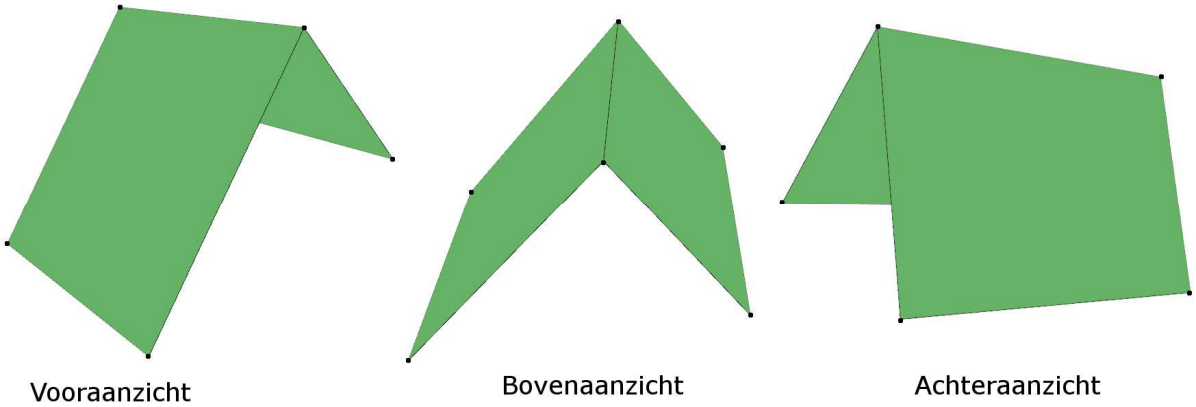
In Figuur 18 staat links de geluidsbron (een punt), hieruit worden drie rays verzonden. Deze drie rays vormen een beam.

Figuur 18: Beams en rays

Binnen een virtueel model definieert een willekeurig drie- of vierhoekig vlak een **element**. Elk element bestaat uit drie of vier nodes en zijn normaal. Een **node** is een x, y, z punt binnen het cartesisch coördinatenstelsel assenstelsel. Een normaal van een element is een eenheidsvector (vector met lengte gelijk aan één) loodrecht op het vlak.

Het vlak waarop een geluidsstraal botst, is het **intersectie element**.

Een model is meestal opgebouwd met samenhangende elementen, deze delen dan één of meerdere nodes. Bijvoorbeeld twee elementen die twee nodes gemeen hebben:



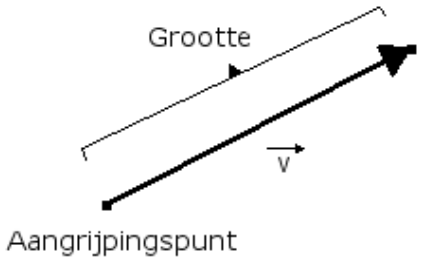
Figuur 19: Elementen met gedeelde grenslijn

De hoek gevormd door deze elementen ten opzichte van de geluidsbron is ook belangrijk voor verdere benaming.
Indien de elementen een hoek maken die kleiner is dan 180° wordt de gedeelde grenslijn **backward wedge** genoemd (zie *Figuur 12: Creeping wave zonder schaduwzone*).
Indien de hoek groter of gelijk is aan 180° wordt de gedeelde grenslijn een **forward wedge** genoemd. (zie *Figuur 11: Creeping wave met schaduwzone*).

Een element dat geen directe buren heeft zal geen grenslijn hebben die wordt gedeeld. Dit element krijgt de term **edge**.

Een microfoon wordt vanaf nu uitgedrukt door de term **fieldpoint**.

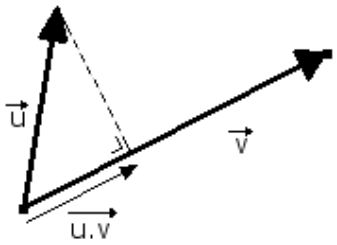
De geometrische akoestiek kan niet bestaan zonder ruimtelijke wiskunde. Het is belangrijk dat de lezer hier een kleine basiskennis van heeft. Het begin van de ruimtelijke wiskunde is een ruimtelijke **vector**. Een vector is een lijn in de ruimte die een grootte, een richting en een aangrijpingspunt heeft.
Het aangrijpingspunt is het beginpunt van de lijn, de grootte is de lengte van de lijn. Samen met de grootte en het aangrijpingspunt bepaalt de richting het eindpunt van de vector.



Figuur 20: Vector

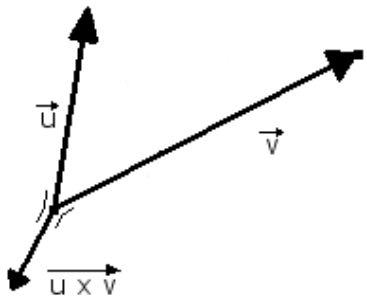
Een **eenheidsvector** is een vector met lengte gelijk aan één. Een vector wordt een eenheidsvector na een normalisatie bewerking.

De belangrijkste bewerkingen met een vector zijn het **dot product** en het **cross product**. De bewerkingen gebeuren op twee vectoren, beide hebben hetzelfde aangrijpingspunt.
Het dot product projecteert een vector op een andere vector.



Figuur 21: Projectie van vector u op vector v

Het cross product berekent de vector die loodrecht op beide gegeven vectoren staat. De richting van de berekende vector wordt bepaald door de rechterhand regel. Bijvoorbeeld het cross product van vector u met vector v: stel de wijsvinger voor als vector u en stel de middenvinger voor als vector v, de duimvinger stelt dan de richting van de nieuwe vector voor.



b. Eerste GA algoritme: Triangular Beam Tracing

De originele Raynoise solver is ontwikkeld met het Triangular Beam Tracing algoritme (TBT).

Een beam wordt gedefinieerd door zijn dragende rays en een middenlijn. De middenlijn wordt gedefinieerd als lijn van de geluidsbron door het incenter punt van de driehoekvormige beam.
Het incenter punt wordt gevonden met behulp van volgende formule:

$$\frac{a(x_a, y_a) + b(x_b, y_b) + c(x_c, y_c)}{P}$$

Figuur 22: Formule incenter van een driehoek

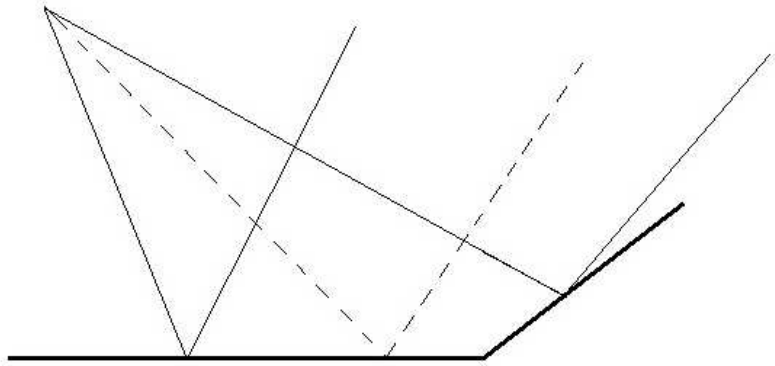
x,y: coördinaten hoekpunten
a: lengte overstaande zijde
P = a + b + c

i. De geluidsbron

De geluidsbron kan eender welke vorm zijn in het TBT algoritme. In functie van wat er wordt gesimuleerd wordt de geluidsbron opgesteld. Als dit een gerichte geluidsbron is, zoals een luidspreker, worden er meerdere beams naast en op elkaar geplaatst. Als de geluidsbron bijvoorbeeld de motor van een auto is, waarbij geluid naar alle mogelijke richtingen wordt verstuurd, zullen de beams als een circulair hoekige vorm worden opgesteld.

ii. Reflectie

De reflectie van een triangular beam wordt gedefinieerd aan de hand van zijn middenlijn. Het element waarop de middenlijn botst wordt gekozen als het intersectie element van de volledige beam. Om het middenpunt te reflecteren is het spiegelpunt van de originele bron nodig. Uit dit spiegelpunt worden eveneens de reflecterende rays bepaald. In *Figuur 23*: *Reflectie Triangular Beam Tracing* is te zien hoe een twee dimensionale triangular beam gereflecteerd wordt. De stippellijn is de middenlijn van de beam die het intersectie element bepaalt. Ten opzichte van dit intersectie element wordt het spiegelpunt van de originele bron gezocht. Dit spiegelpunt is de bron van beide reflecterende rays.



Figuur 23: Reflectie Triangular Beam Tracing

Het valt onmiddellijk op dat de reflectie van de meest rechtse ray niet correct gebeurt. Het echte intersectie element van deze ray wordt genegeerd omdat de reflectie bij een triangular beam gedefinieerd is door zijn middenlijn.

iii. Diffractie

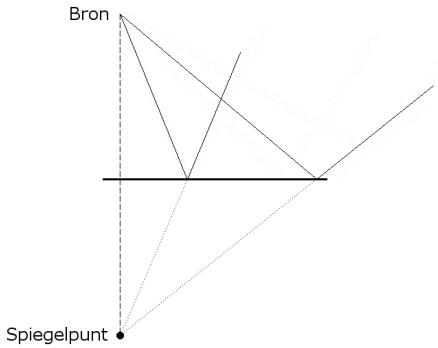
Diffractie wordt niet specifiek besproken in TBT algoritme. Een mogelijkheid om diffractie toch te implementeren is door de eventuele diffractie punten te specificeren binnen het virtuele model. Deze punten worden dan als nieuwe geluidsbronnen bekeken.

c. Tweede GA algoritme: Adaptive Beam Tracing

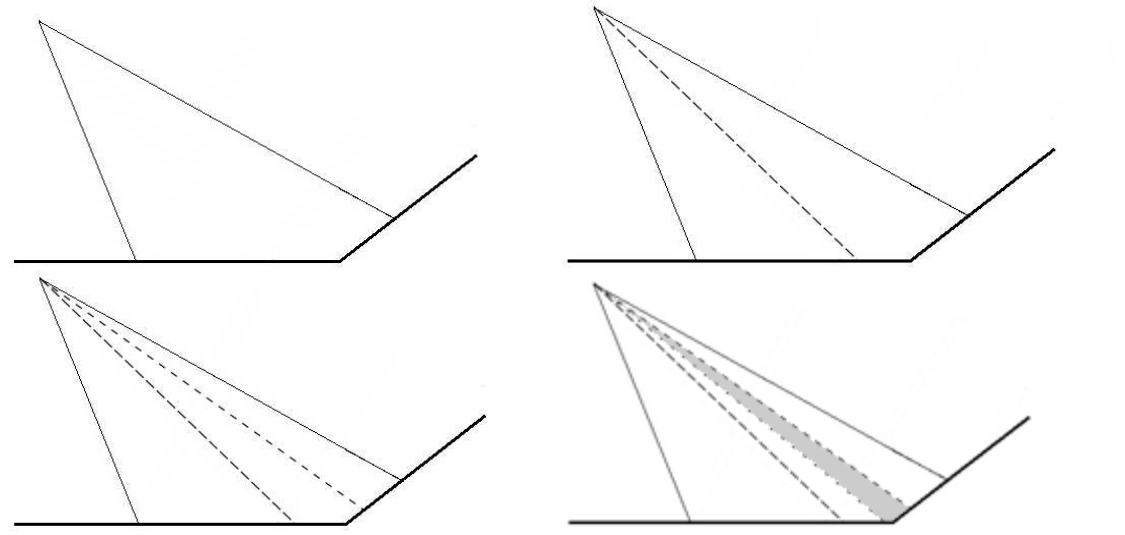
Het Adaptive Beam Tracing algoritme werd eerst voorgesteld door I. A. Drumm en Y. W. Lam in de universiteit van Salford (UK) op 21/06/1999 (Bibliografie [8]).

Adaptive Beam Tracing is oorspronkelijk afgeleid uit Triangular Beam Tracing. Het TBT algoritme is daarom ook ongeveer vergelijkbaar met het ABT algoritme. Het enige, maar zeer belangrijke, verschil is dat de beams bij Adaptive Beam Tracing adaptive zijn met het virtueel model. Adaptive betekent "aanpasbaar", Adaptive beams zullen zich aanpassen aan het element waar ze op botsen.

Ten opzichte van het TBT algoritme wordt er niet meer gewerkt met een middenlijn, het ABT algoritme definieert de beam enkel door zijn dragende rays. **Elke ray wordt afzonderlijk gereflecteerd vanuit zijn eigen spiegelpunt**.



Neem het voorbeeld dat gegeven werd bij Triangular Beam Tracing (*Figuur 23: Reflectie Triangular Beam Tracing*).
De twee rays vallen niet op hetzelfde element. Het ABT algoritme definieert dat de invallende beam moet worden opgesplitst.
De eerste opdeling splitst de originele beam in twee. Eén beam botst volledig op één element, de andere botst op beide elementen, deze wordt nogmaals opgesplitst. Dit blijft zich herhalen tot een bepaald maximum.



Figuur 24: ABT opsplitsingen

Op Figuur 24 is te zien dat na de laatste opsplitsing er drie beams zijn die volledig op één element botsen en er nog steeds één beam is die niet volledig op één element botst, een verdere opdeling zal hier ook niets aan verhelpen. Deze laatste beam zal onderhevig zijn aan backward diffractie.

i. Geluidsbron

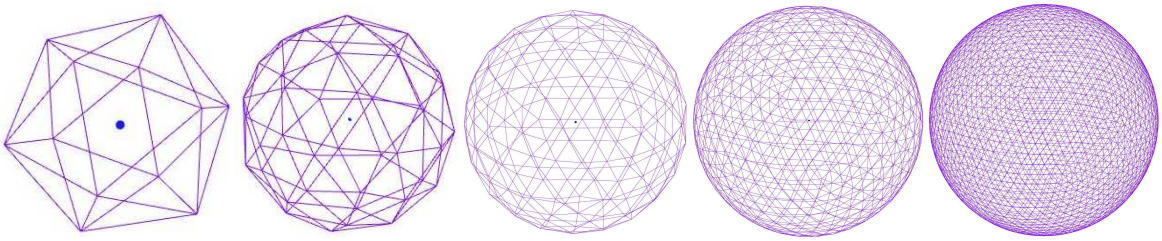
Het oorspronkelijke Adaptive Beam Tracing algoritme definieert geluidsbronnen die in alle richtingen evenwaardige beams verzenden. Dit zijn beams die dezelfde intensiteit en oppervlakte hebben. Een dergelijke geluidsbron wordt gesimuleerd door een icosahedron. Een icosahedron is een ruimtelijk figuur met 20 driehoekige vlakken, 30 ribben en 12 hoekpunten.
De figuur wordt gedefinieerd aan de hand van de combinaties van volgende coördinaten, 12 in totaal:
$(0, \pm 1, \pm \varphi)$
$(\pm 1, \pm \varphi, 0)$
$(\pm \varphi, 0, \pm 1)$
,waar φ = (golden ratio) = $(1+\sqrt{5})/2$.

Elke ray van de bron naar 1 van bovenstaande coördinaten is een hoekpunt van aanliggende beams. Een ray wordt door verschillende beams gedeeld.

Een belangrijk nadeel van deze methode is dat de oppervlakte van een beam steeds groter wordt naarmate de beam zich verder van de geluidsbron verwijderd. Dit is een cruciaal nadeel voor de reflectie van de beam (zie later: *ii. Reflectie*).
Om dit probleem op te lossen, kan de geluidsbron in meerdere beams opgedeeld worden. Een beam wordt opgedeeld in 4 kleinere beams.

Figuur 25 stelt de opsplitsing van een geluidsbron voor.

Links op de afbeelding staat er een standaard icosahedron met 12 beams. Een niveau verder wordt elke beam opgedeeld in 4 beams zodat er 48 beams ontstaan. Op de figuur worden ook de volgende stadia van de opsplitsende beams weergegeven. De meest rechtse geluidsbron, zichtbaar op de afbeelding, bevat 3072 ($=12*(4^4)$) beams.



Figuur 25: Opsplitsing geluidsbron

De opsplitsing van een beam wordt verder besproken in hoofdstuk *10.c.iii De ABT sterkte: beams opsplitsen*.

Een bolvormige geluidsbron is niet altijd gewenst. Stel bijvoorbeeld dat we de luidspreker van een auto willen simuleren. Een luidspreker is een gerichte geluidsbron, het is zeer onrealistisch dat hiervoor een icosahedron wordt gebruikt. Daarom is er beslist dat er ook een mogelijkheid moet zijn om een zelf gedefinieerde geluidsbron te implementeren. Dit kan een halve icosahedron zijn, maar zelfs ook één beam. De zelf gedefinieerde geluidsbronnen kunnen nog steeds verder worden opgesplitst.

ii. Reflectie

1) Intersectie punten

Voor een beam kan gereflecteerd worden, moeten eerst de intersectiepunten van zijn dragende rays gekend zijn. Het intersectie punt van een ray en een element is het punt waar de ray op het vlak botst. Dit intersectie punt wordt dan de bron van de gereflecteerde ray.

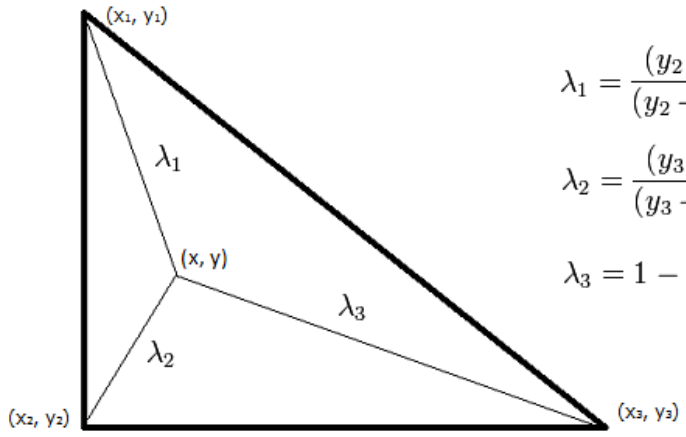
Het intersectie punt van een ray en een element wordt berekend met behulp van enkele logische vector bewerkingen.

Eerst wordt er gecontroleerd of de ray in de juiste richting ligt om met het element te botsen. Dit gebeurt door de afstand van de bron van de ray tot het element te berekenen. Als deze afstand negatief is, ligt het element achter de ray en zal de ray dus nooit tegen dit element botsen.

Om de afstand te berekenen zijn er 2 vectoren nodig.
De eerste vector wordt aangemaakt tussen de bron van de ray en een willekeurig punt op het element. De tweede vector is de normaal van het element in de richting van de bron van de ray.
De eerste vector, geprojecteerd op de tweede vector bepaalt de loodrechte afstand tussen de bron van de ray en het element.

Het eventuele intersectiepunt van de ray op de driehoek is dan gelijk aan de richting van de ray, die een eenheidsvector is, vermenigvuldigd met berekende afstand opgeteld bij de bron van de ray.
ip = source + dir * dist;

Vervolgens wordt er getest of het intersectie punt binnen de grenzen van een element ligt. Dit kan met behulp van de barycentrische coördinaten van het intersectie punt. De barycentrische coördinaten definiëren een punt ten opzichte van de hoekpunten van een driehoek.



$$\lambda_1 = \frac{(y_2 - y_3)(x - x_3) + (x_3 - x_2)(y - y_3)}{(y_2 - y_3)(x_1 - x_3) + (x_3 - x_2)(y_1 - y_3)}$$

$$\lambda_2 = \frac{(y_3 - y_1)(x - x_3) + (x_1 - x_3)(y - y_3)}{(y_3 - y_1)(x_2 - x_3) + (x_1 - x_3)(y_2 - y_3)}$$

$$\lambda_3 = 1 - \lambda_1 - \lambda_2 \, .$$

Figuur 26: Barycentrische coördinaten

Een punt ligt in een driehoek als $0 \leq \lambda_i \leq 1 \; \forall \; i \; \text{in} \; 1, 2, 3$.

Opmerking: de barycentrische formule kan enkel gebruikt worden in een tweedimensionaal assenstelsel. Voor virtualisatie van akoestiek wordt er uiteraard een driedimensionaal assenstelsel gebruikt. Daarom moeten het element en de ray eerst geprojecteerd worden op het meest bepalende tweedimensionaal assenstelsel van het element.

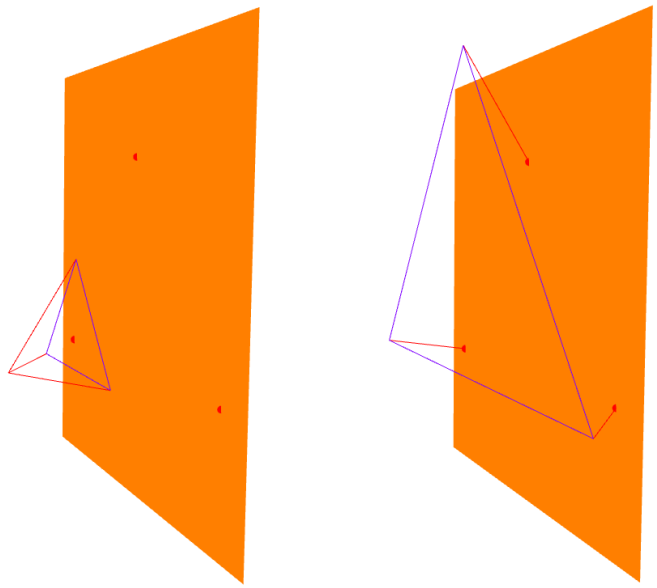
2) De reflectie

Nu de intersectie punten van alle rays gevonden zijn kunnen de rays gereflecteerd worden.
Het berekende intersectie punt van een ray vormt de bron van de nieuw reflecterende ray. De richting van de ray wordt bepaald met behulp van het spiegelpunt van de originele bron. Hiervoor wordt de bron gespiegeld over het intersectie element.

Opnieuw is de loodrechte afstand tussen de originele bron van de ray en het botsende element nodig (zie vorig hoofdstuk: *10.c.ii Intersectie punten*).

Stel:
d = een vector die de loodrechte afstand voorstelt, gericht naar het element
src = originele bron van de invallende beam
Dan wordt het spiegelpunt als volgt berekend:
sPunt = src + d * 2

Nu alle botsende rays gereflecteerd zijn, zijn ook alle reflecterende beams gedefinieerd, aangezien een beam gedefinieerd is door zijn dragende rays.



Figuur 27: reflectie, zijaanzicht

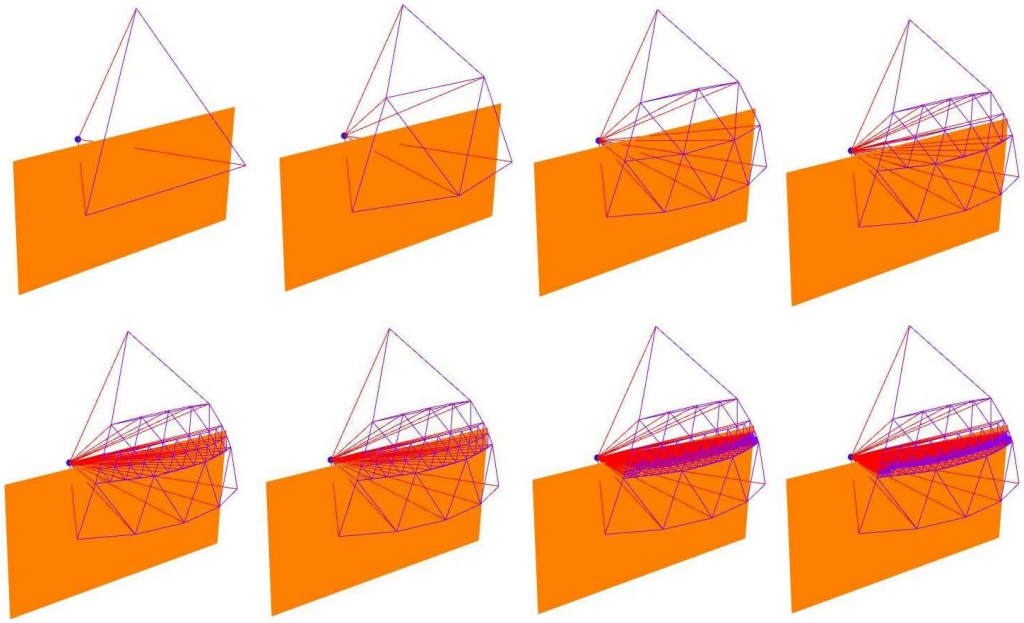
Figuur 27 toont het zijaanzicht van een botsende beam. De linker figuur toont de invallende beam en de intersectie punten van zijn rays. De rechter figuur toont de gereflecteerde beam.

iii. De ABT sterkte: beams opsplitsen

Geometrical Acoustics maakt gebruik van rays om beams te reflecteren. Maar wat als de dragende rays van een beam niet op hetzelfde element botsen?

Hier komt de sterkte van Adaptive Beam Tracing tot zijn werking. Een dergelijk beam wordt opgesplitst tot kleinere beams. Dit zal zich blijven herhalen tot het aantal opsplitsingen een (vooraf gedefinieerd) maximum heeft bereikt.

De opsplitsing van de beam gebeurt op dezelfde manier als bij de opsplitsing van de geluidsbron, alleen worden hier specifieke beams opgesplitst en niet alle beams. Een beam wordt opgedeeld in vier kleinere beams. Hiervoor zijn drie nieuwe rays nodig, elk in het midden van de verbinding tussen twee bestaande rays.



Figuur 28: Beam opsplitsen

Figuur 28 verduidelijkt deze opsplitsing. De beam links boven is de originele beam die werd verzonden uit een geluidsbron. Hierbij botsen de twee onderste rays met een element, de bovenste ray botst niet en loopt door tot in de oneindigheid.

Aangezien niet alle rays op hetzelfde element botsen wordt de beam opgesplitst. Er worden vier nieuwe beams gevormd. De bovenste beam heeft drie rays die tot in de oneindigheid worden verzonden, deze beam hoeft niet verder te worden opgesplitst. De onderste drie beams hebben steeds één of twee rays die niet op het element botsen, deze beams worden elk opgesplitst.

Deze stap herhaalt zich tot een maximum van opsplitsingen is bereikt. Beams die niet op hetzelfde element botsen worden voorlopig genegeerd (deze beams worden gediffracteerd, zie hoofdstuk: *10.c.iv Diffractie: astigmatische beams en Keller Cones*).

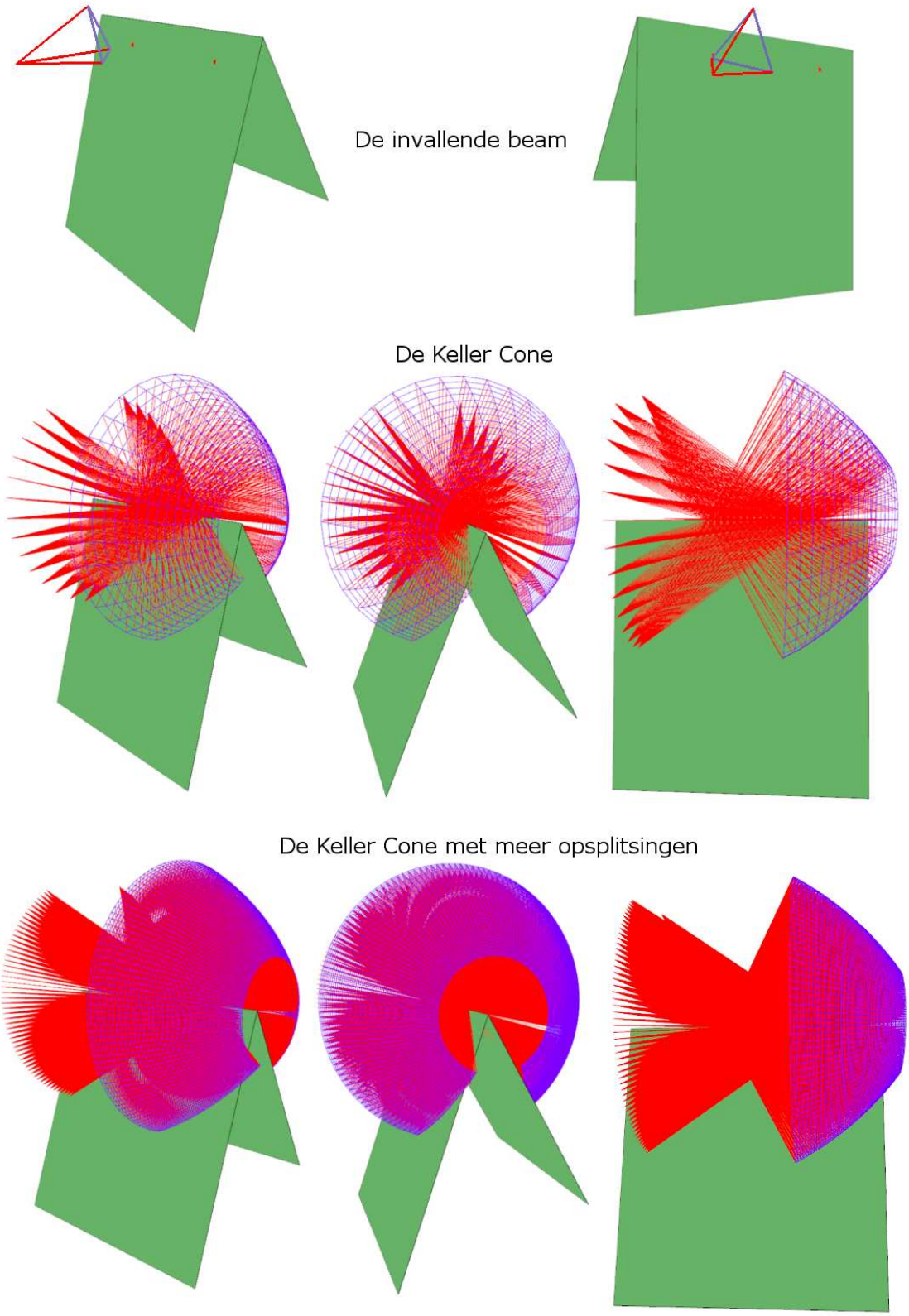
Het aantal opsplitsingen is hier van cruciaal belang.
Een groot aantal opsplitsingen zorgt voor veel kleine beams die veel geheugen gebruiken.
Een klein aantal opsplitsingen zorgt ervoor dat sommige beams verkeerd worden behandeld. Neem bijvoorbeeld Figuur 28, indien de opsplitsing gedaan was na één opsplitsing (met vier beams in totaal) zou er geen reflectie ontstaan omdat er geen enkele beam volledig op een element botst. Deze beperking wordt later verder besproken in hoofdstuk *11.c.i Opsplitsen van een invallende beam.*

iv. Diffractie: astigmatische beams en Keller Cones

In het vorige hoofdstuk werd de manier besproken waarop beams worden opgesplitst. Een beam wordt opgesplitst tot kleinere beams tot het aantal opsplitsingen een vooraf gedefinieerd maximum heeft bereikt. Wanneer een beam maximaal is opgesplitst en er nog steeds kleinere beams overblijven die niet op hetzelfde element botsen worden deze rond het intersectie element gediffracteerd.

Diffractie wordt in de Geometrical Acoustics voorgesteld door de *geometrical theory of diffraction (GTD)*. GTD werd oorspronkelijk bedacht door Joseph Keller in 1962 (Bibliografie [17]).

Een geluidsgolf die botst op de grenslijn van twee elementen zorgt voor een nieuwe bron op de grenslijn. De nieuwe bron zal dan op zijn beurt nieuwe golven uitzenden (zie *9.a.v Diffractie, gebaseerd op de Geometrical Theory of Diffraction*).
Adaptive Beam Tracing maakt gebruik van beams, dus zullen er nieuwe beams moeten worden gevormd rond de grens van twee elementen. De GTD definieert deze nieuwe bron als *de Keller Cone*.

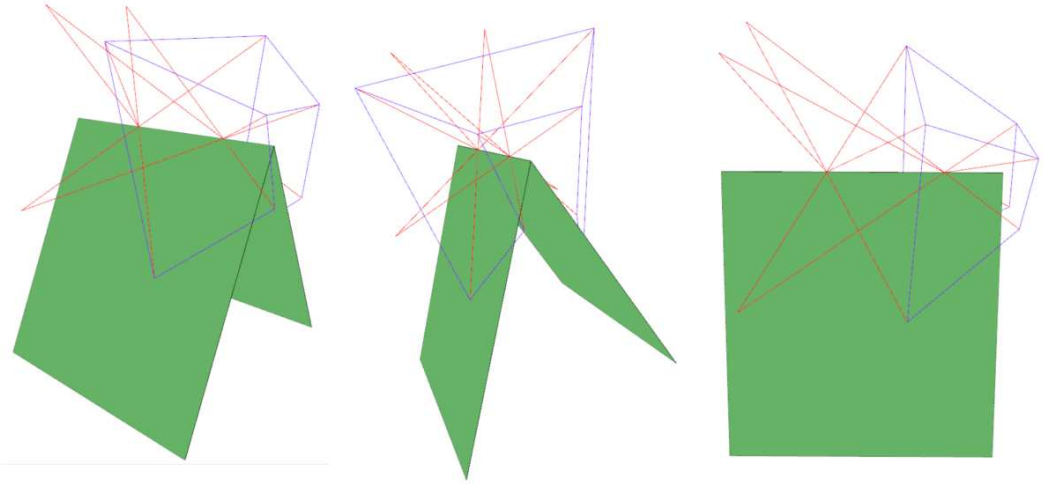


Figuur 29: Introductie van de Keller Cone

De afbeelding hierboven illustreert een Keller Cone op een forward wedge.
Bovenaan is de invallende beam getekend, deze botst niet volledig op het element want zijn bovenste ray gaat over het element naar de oneindigheid. Om de Keller Cone te verduidelijken wordt de invallende beam niet verder opgesplitst, de invallende beam wordt dan gediffracteerd en zal een Keller Cone maken. De Keller Cone zorgt ervoor dat er beams worden verstuurd in alle richtingen rondom de grenslijn tussen de twee elementen.

Wat opvalt, is dat de verzonden beams van de Keller Cone anders zijn dan de standaard driehoekige beams. Bij de Keller Cone techniek worden **astigmatische beams** verzonden. Dit zijn vierhoekige beams die een bronlijn hebben, in tegenstelling tot de standaard driehoekige beams die uit een puntbron ontstaan.

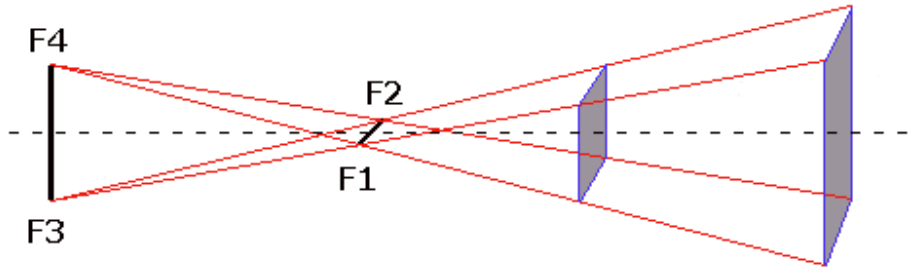
De standaard Keller Cone bestaat uit drie astigmatische beams. Voor verdere nauwkeurigheid kan de standaard Keller Cone worden opgesplitst. De opsplitsingen gebeuren steeds op een standaard Keller Cone.



Figuur 30: Introductie standaard Keller Cone

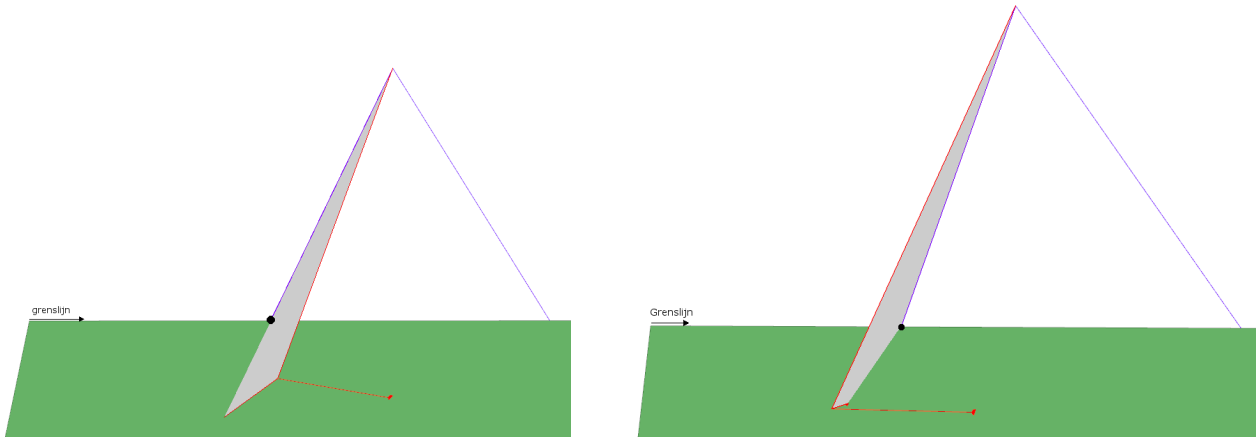
Hierop volgt hoe een Keller Cone wordt opgemaakt met astigmatische beams. Om het overzicht te behouden zal de verdere uitleg gebruik maken van een standaard Keller Cone, dus een Keller Cone zonder opsplitsingen. Later wordt besproken hoe de standaard Keller Cone wordt opgesplitst.

1) De opbouw van een astigmatische beam



Figuur 31: De astigmatische beam

Een Keller Cone is gevormd door astigmatische beams, die op hun beurt gedefinieerd zijn door vier rays. Deze rays ontstaan door een combinatie van hoekpunten van 2 lijnen (in Figuur 31: De astigmatische beam F1-F2 en F3-F4).
De eerste lijn is de reflectielijn (in *Figuur 31: De astigmatische* beam: F1, F2), het is een deel van de gedeelde grenslijn tussen twee elementen waar een beam wordt rond gediffracteerd.

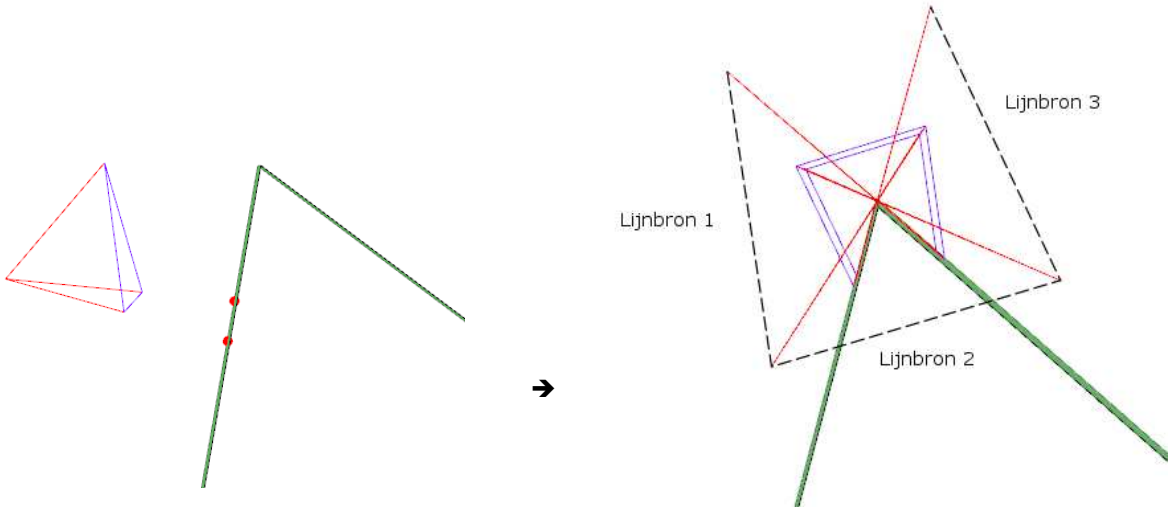


Figuur 32: draagvlak beam

De reflectielijn wordt gevonden aan de hand van dezelfde intersectiemethode die gebruikt werd bij het zoeken naar intersectiepunten van rays en element.
Een vector definieert de grenslijn van een element, met als beginpunt een grenspunt en als eindpunt een aanliggend grenspunt van het element. Verder wordt een dragend vlak van een beam gedefinieerd als nieuw element. Een normale beam heeft drie draagvlakken, waarvan er maximaal en minimaal twee zijn die de grenslijn definiëren. Tussen de grenslijn vector en een dragend element wordt het intersectiepunt berekend, zie *Figuur 32: draagvlak beam* .

Dezelfde berekening wordt uitgevoerd op het andere dragend vlak. Opnieuw wordt het intersectiepunt door middel van enkele vectorberekeningen en de barycentrische methode gevonden (*10.c.ii. Intersectie punten*).

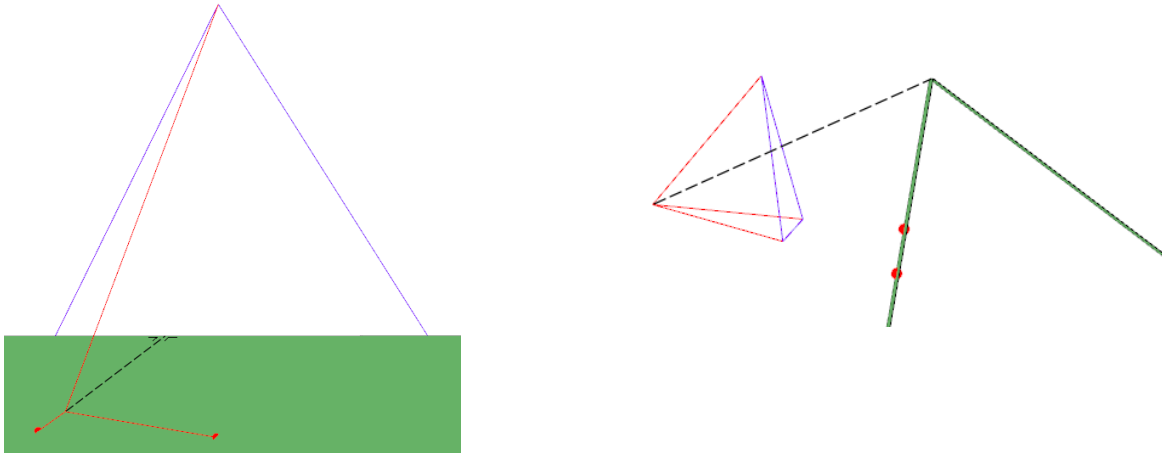
Een standaard Keller Cone heeft één reflectielijn, voor verdere nauwkeurigheid wordt de reflectielijn opgesplitst in meerdere kleinere lijnen.
De nauwkeurigheid van een Keller Cone wordt later besproken.



Figuur 33: Bronlijnen Keller Cone

De tweede lijn om een astigmatic beam te maken **is de bronlijn**. Op Figuur 33 is te zien dat de invallende beam rondom het hoekig element gediffracteerd wordt. Om dit circulaire effect te bereiken zijn er verschillende bronnen nodig, allemaal op een zelfde afstand van de reflectielijn. De bron wordt hierbij een lijn, wat noodzakelijk is om een vierhoekige beam te verkrijgen. Alle bronlijnen achter elkaar geplaatst vormen samen een hoekige cirkelvorm rondom de reflectielijn. Om de bronlijnen te bepalen zijn er **twee variabelen** nodig: **de afstand tussen de bron en de grenslijn** en **de hoek tussen de grenzende elementen**.

Afstand tussen de bron en de grenslijn:

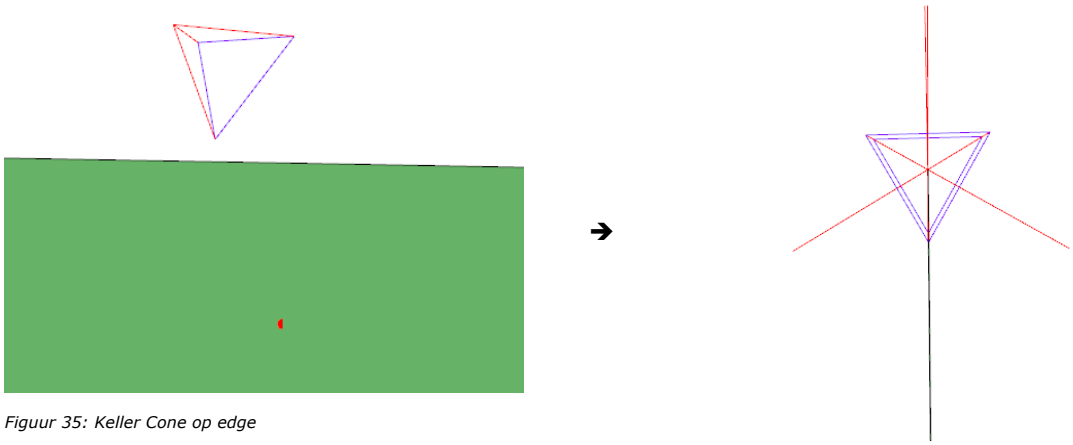


Figuur 34: Loodrechte afstand tussen bron en grenslijn

Vector a heeft als aangrijpingspunt een willekeurig punt op de grenslijn en als richting de bron van de beam.
Vector b is het cross product van de normaal van het element en de grenslijn.
De projectie van vector a op vector b geeft de loodrechte afstand van de bron tot de grenslijn.
De projectie bekomt men door het dot product te nemen van beide vectoren.
Zo kent men ook de bron loodrecht op de grenslijn, deze is belangrijk om de bronlijnen op een correcte plaats rond de grenslijn te roteren.

2) De opbouw van de Keller Cone

Indien een beam op een grenslijn botst van een alleenstaand element, ontstaat er een Keller Cone 360° rond de grenslijn.



Figuur 35: Keller Cone op edge

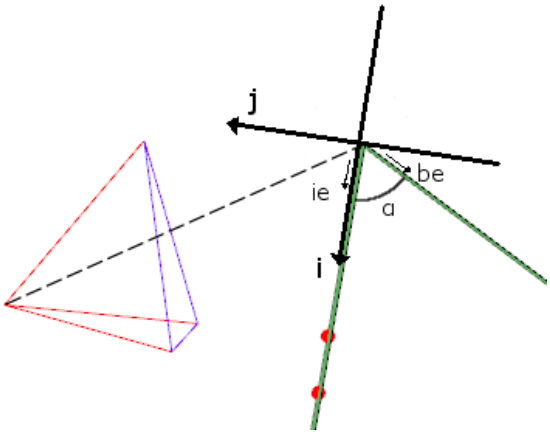
Indien de Keller Cone op een gedeelde grenslijn ontstaat, moet de hoek tussen de grenzende elementen berekend worden (zie *Figuur 33: Bronlijnen Keller Cone*).

Hiervoor zijn volgende vectoren nodig:
Vector n1: de normaal vector van het intersectie element, in de richting van de bron.
Vector n2: de normaal vector van het buur element.
Vector boundVec: een vector op de grenslijn.

Vector defLineIntElem: definitielijn van het intersectie element
Vector defLineNeighbElem: definitielijn van het buur element
Beide definitielijnen worden berekend met het cross product van hun respectievelijke normaal en de vector die de grenslijn definieert.
defLineIntElem = n1 X boundVec
defLineNeighbElem = n2 X boundVec

Zie *Figuur 36*: vectoren ie en be stellen beide de definitielijn van het intersectie – en het buur element voor.

Nu de definitielijnen bepaald zijn, kan de hoek van de Keller Cone berekend worden. Het dot product van beide definitielijnen geeft de projectie van één lijn op de andere. De inverse cosinus van het dot product geeft de kleinste hoek tussen de twee definitielijnen.



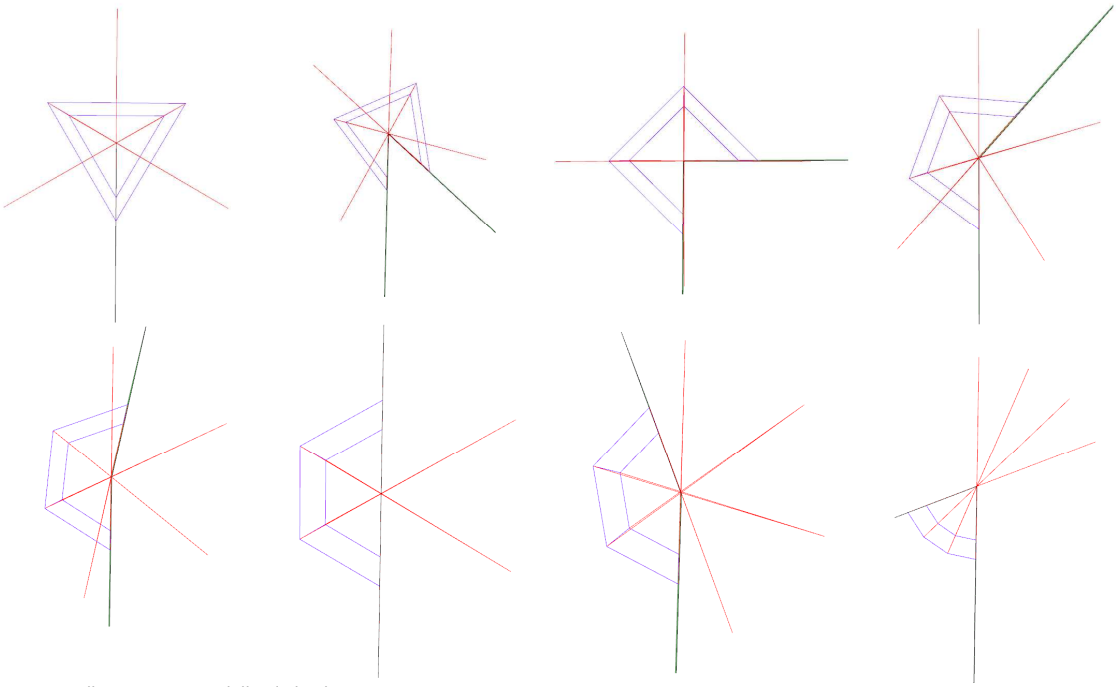
Figuur 36: Hoek tussen de grenzende elementen

De kleinste hoek (α) is niet altijd de juiste oplossing, zoals het bovenstaande voorbeeld. Daarom is het nodig om een extra controle te doen om te zien hoe beide definitielijnen liggen ten opzichte van elkaar. Hiervoor is een virtueel assenstelsel nodig:
x-as: de definitielijn van het intersectie element (i)
y-as: de normaal van het intersectie element (j)

Met de projectie van de definitielijn van het buur element op zowel de x-as als de y-as wordt er berekend in welk kwadrant het buur element zich bevindt.

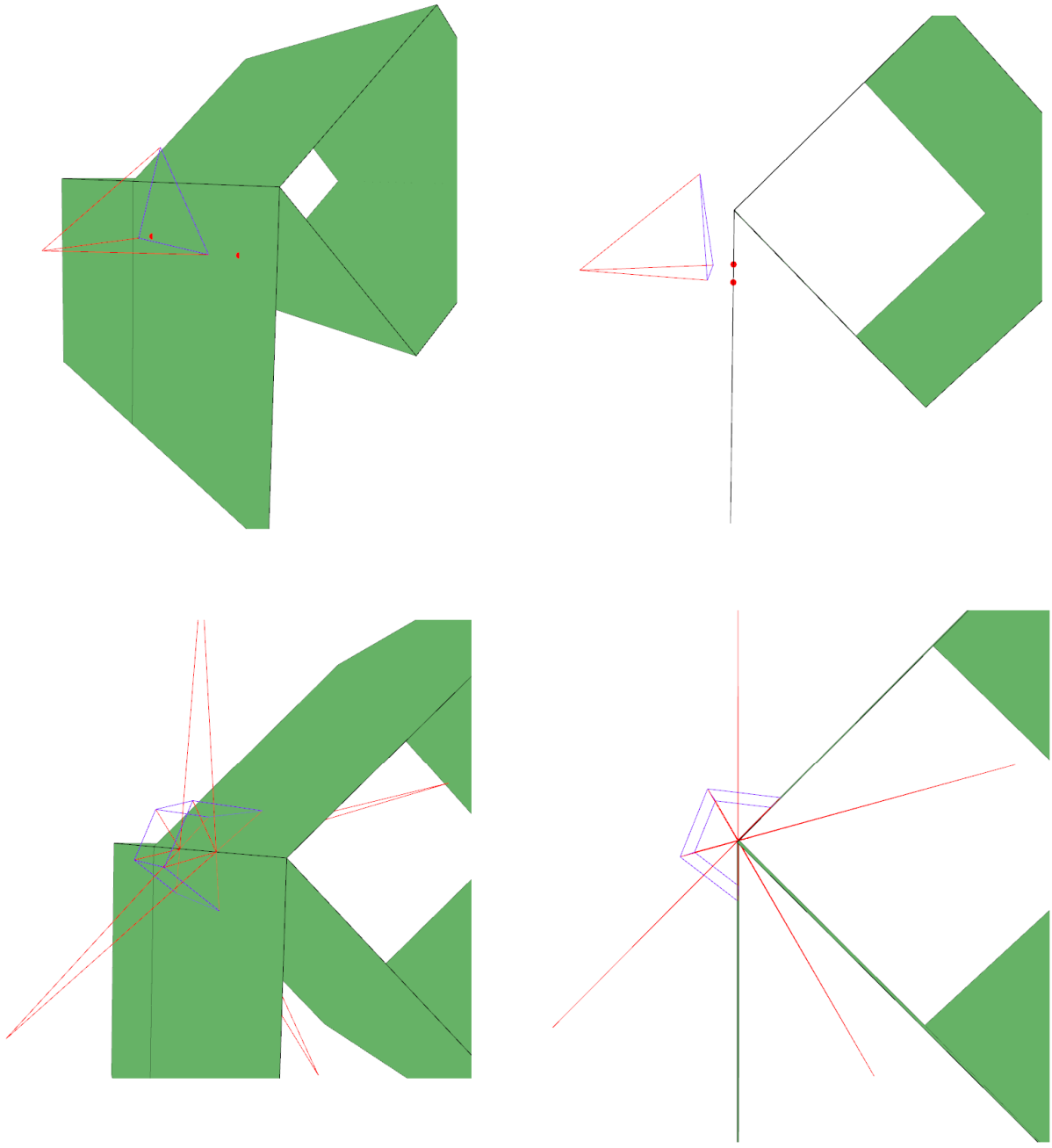
Als de x-as projectie < 0 en de y-as projectie < 0 dan bevindt het buur element zich in het 3de kwadrant, zodat de hoek tussen de elementen gelijk is aan 360° – de berekende kleinste hoek.

Als de x-as projectie > 0 en de y-as projectie < 0 dan bevindt het buur element zich in het 4de kwadrant, zodat de hoek tussen de elementen ook gelijk is aan 360° – de berekende kleinste hoek.



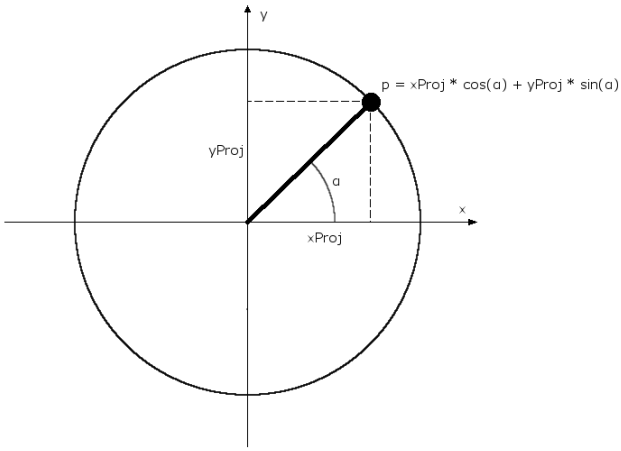
Figuur 37: Keller Cone in verschillende hoeken

Indien de grenslijn gedeeld wordt door meer dan twee elementen moet deze methode meerdere malen worden toegepast. De kleinst gevonden hoek tussen elk paar definitielijnen, bepaalt dan het paar waarop de Keller Cone wordt gemaakt.



Figuur 38: Keller Cone op triple gedeelde grenslijn

De correcte hoek tussen de definitielijnen is nu gekend, deze hoek wordt verder opgesplitst om de eindpunten van de bronlijnen te bepalen. Bij een standaard Keller Cone wordt de hoek opgesplitst in drie kleinere hoeken (subHoek), er worden dus drie bronlijnen gemaakt.
Bijvoorbeeld: als de hoek tussen twee naburige elementen 120° is, zullen de opgedeelde hoeken 40° zijn (120°/3).
De opsplitsing gebeurt opnieuw met behulp van een virtueel assenstelsel met op de x-as de definitielijn van het intersectie element (i) en op de y-as de normaal van het intersectie element (j). Op dit assenstel wordt de sinus, cosinus definitie van een cirkel toegepast zodat de eindpunten van de bronlijnen kunnen worden berekend.



Figuur 39: Sin- cosinus definitie cirkel

Toegepast op de bronlijnen geeft dit:
Punt a = i*cos(subHoek) + j*sin(subHoek)

De reflectielijnen en bronlijnen zijn gekend, de beams kunnen worden opgesteld. Elke astigmatische beam wordt gedefinieerd door vier rays. De rays zijn een combinatie van de twee eindpunten van de bronlijn en van de reflectielijn (zie *Figuur 31: De astigmatische* beam).

3) Akoestische eigenschappen van een astigmatische beam

De astigmatische beams zijn nog steeds "adaptive", toch hebben ze als gevolg dat sommige vooraf besproken technieken en akoestische eigenschappen moeten worden aangepast.

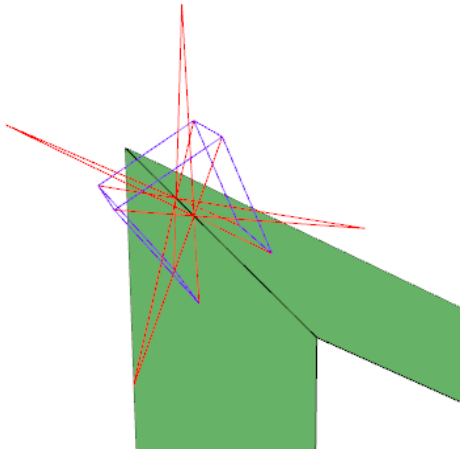
De **reflectie** van een astigmatische beam gebeurt nog steeds met behulp van zijn dragende rays. Voor de reflectie zijn de reflectielijn en bronlijn van een beam niet meer nodig. Elke dragende ray heeft zijn eigen puntbron; deze puntbronnen worden opnieuw gespiegeld. De spiegelpunten vormen dan opnieuw de bronlijnen van de nieuwe astigmatische beam.

De **opsplitsing** van een astigmatische beam gebeurt nog steeds door de beam in vier even grote beams op te splitsen. De afstand tussen elk aanliggend paar dragende rays wordt in twee gedeeld, dit is het richtingspunt van een nieuwe ray. De bron van de nieuwe ray wordt ook bepaald door het middenpunt van de bronlijn, behorende tot de originele rays, te berekenen.

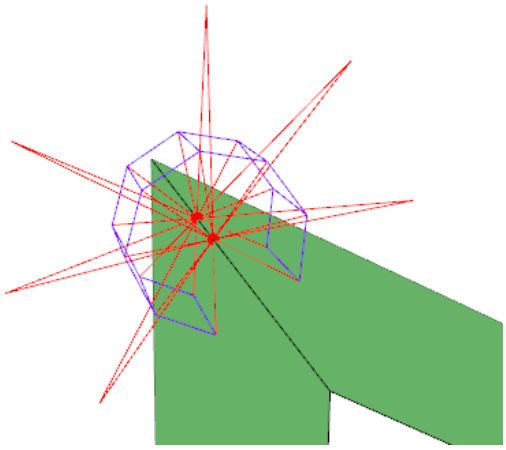
Onderzoeken of een **fieldpoint in een astigmatische beam** ligt gebeurt op exact dezelfde manier als die bij de driehoekige beams, nu met vier vector normaal berekeningen in plaats van drie.

4) Grotere nauwkeurigheid door opsplitsen van astigmatische beams

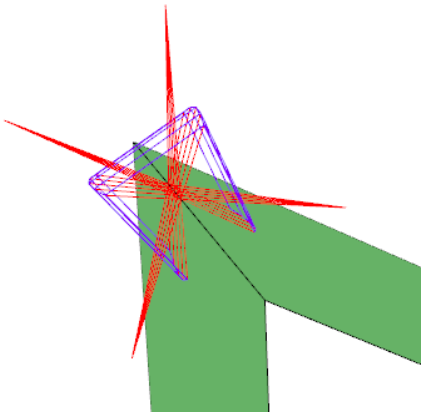
De standaard Keller Cone kan worden opgesplitst naar meer dan drie astigmatische beams. De opsplitsing gebeurt door de bronlijnen op te delen (*Figuur 40*) of door de reflectielijn op te delen (*Figuur 43* en *Figuur 42*) of door beide opsplitsingen (*Figuur 45* en *Figuur 44*).



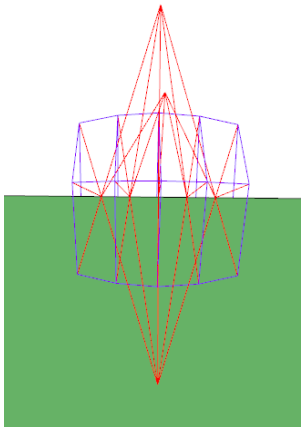
Figuur 41: Standaard Keller Cone



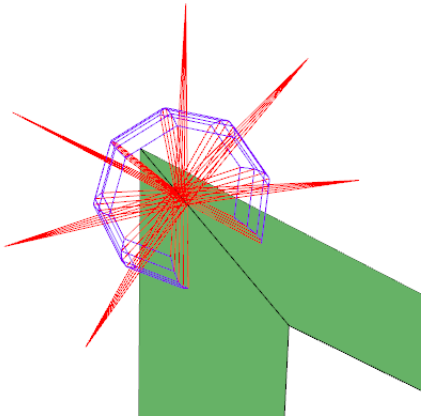
Figuur 40: Keller Cone met bronlijn opsplitsing



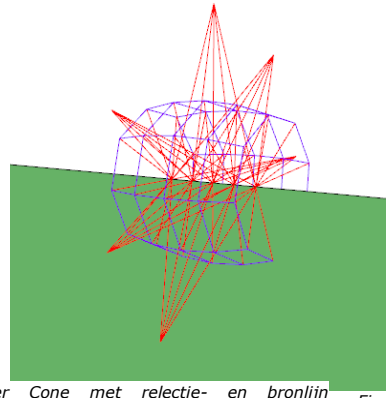
Figuur 43: Keller Cone met reflectielijn opsplitsing, zijaanzicht



Figuur 42: Keller Cone met reflectielijn opsplitsing, vooraanzicht



Figuur 45: Keller Cone met relectie- en bronlijn opsplitsingen, zijaanzicht



Figuur 44: Keller Cone met reflectie- en bronlijn opsplitsingen, vooraanzicht

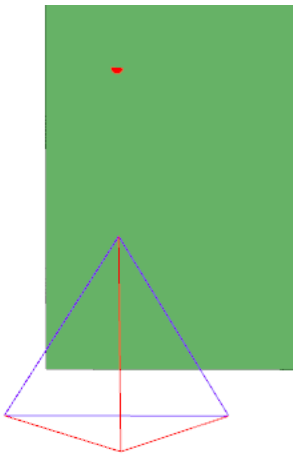
Net zoals bij de opsplitsing van gewone beams die botst op een grenslijn, is het geheugen gebruik bij opsplitsing van een Keller Cone ook van groot belang. Dit wordt later besproken in hoofdstuk *11.c.ii Onderverdeling van een Keller Cone*.

5) Uitzonderingen

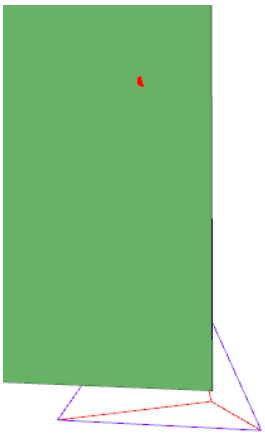
- **Node in invallende beam**

Het is mogelijk dat een beam (na opdeling) op één of meerdere hoeken valt van een element. In functie van het aantal hoeken in de beam zullen er Keller Cones gegenereerd worden op meerdere grenslijnen van hetzelfde element. Hieronder wordt een voorbeeld gegeven met een driehoekige invallende beam. Het is ook mogelijk dat een astigmatische beam op meerdere grenslijnen valt, dezelfde methode wordt dan toegepast.

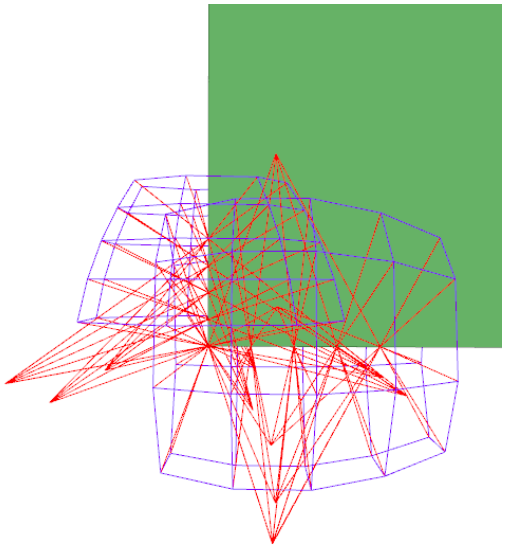
Stel dat een beam op twee grenslijnen van een element valt.



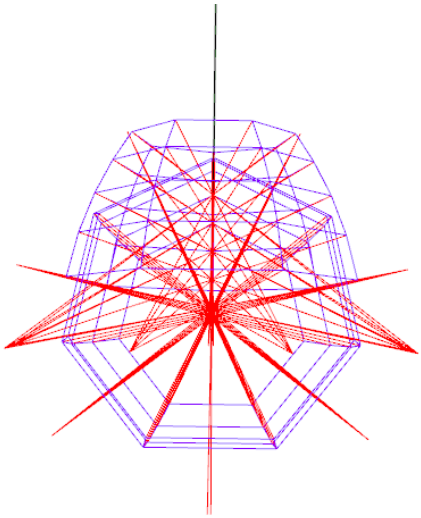
Figuur 47: Beam op twee grenslijnen, vooraanzicht



Figuur 46: Beam op twee grenslijnen, achteraanzicht



Figuur 48: Keller Cone op twee grenslijnen, vooraanzicht

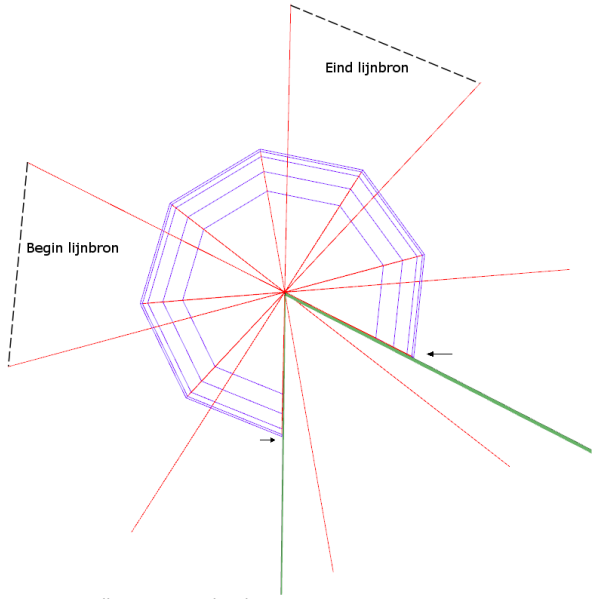


Figuur 49: Keller Cone op twee grenslijnen, zijaanzicht

De intersectie punten van de dragende vlakken van de beam en de grenslijnen van een element worden op dezelfde manier als hierboven berekend. De methode om een hoekpunt in een beam te detecteren is dezelfde als die gebruikt wordt om een fieldpoint te vinden.

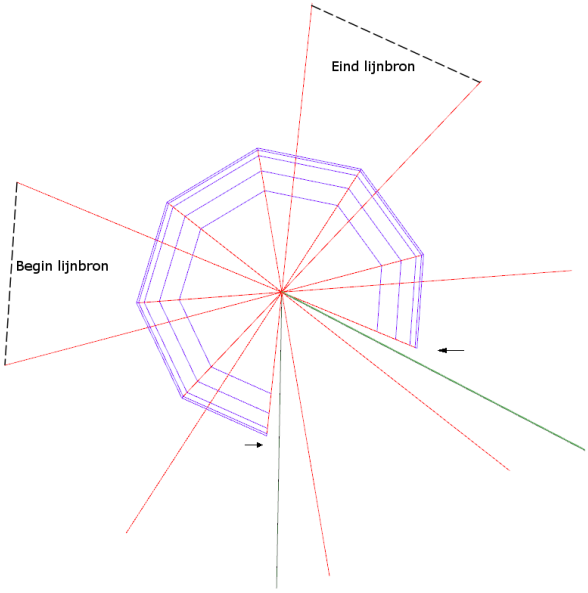
- **Deviatie toepassing op begin- en eind bronlijn.**

Er is gezegd dat de hoek tussen naburige elementen wordt opgesplitst naar mate het aantal bronlijnen die nodig zijn om de Keller Cone te maken. Met een exacte verdeling van de hoek zullen er enkele rays op één of twee naburige elementen staan. Wat als gevolg heeft dat de beam die een dergelijke ray gebruikt niet verder zal geraken dan de oppervlakte van het element, omdat de ray letterlijk in het element ligt en onmiddellijk opnieuw met het element botst, zie *Figuur 50: Keller Cone zonder deviatie .*



Figuur 50: Keller Cone zonder deviatie

Omwille van deze reden krijgen de rays die op een element zouden liggen een kleine deviatie. Deze deviatie wordt dus enkel toegepast op de begin- en de eind bronlijn van een Keller Cone. Het beginpunt van de begin bronlijn staat hoger en het eindpunt van de eind bronlijn staat lager, zie *Figuur 51: Keller Cone met (overdreven) deviatie .*

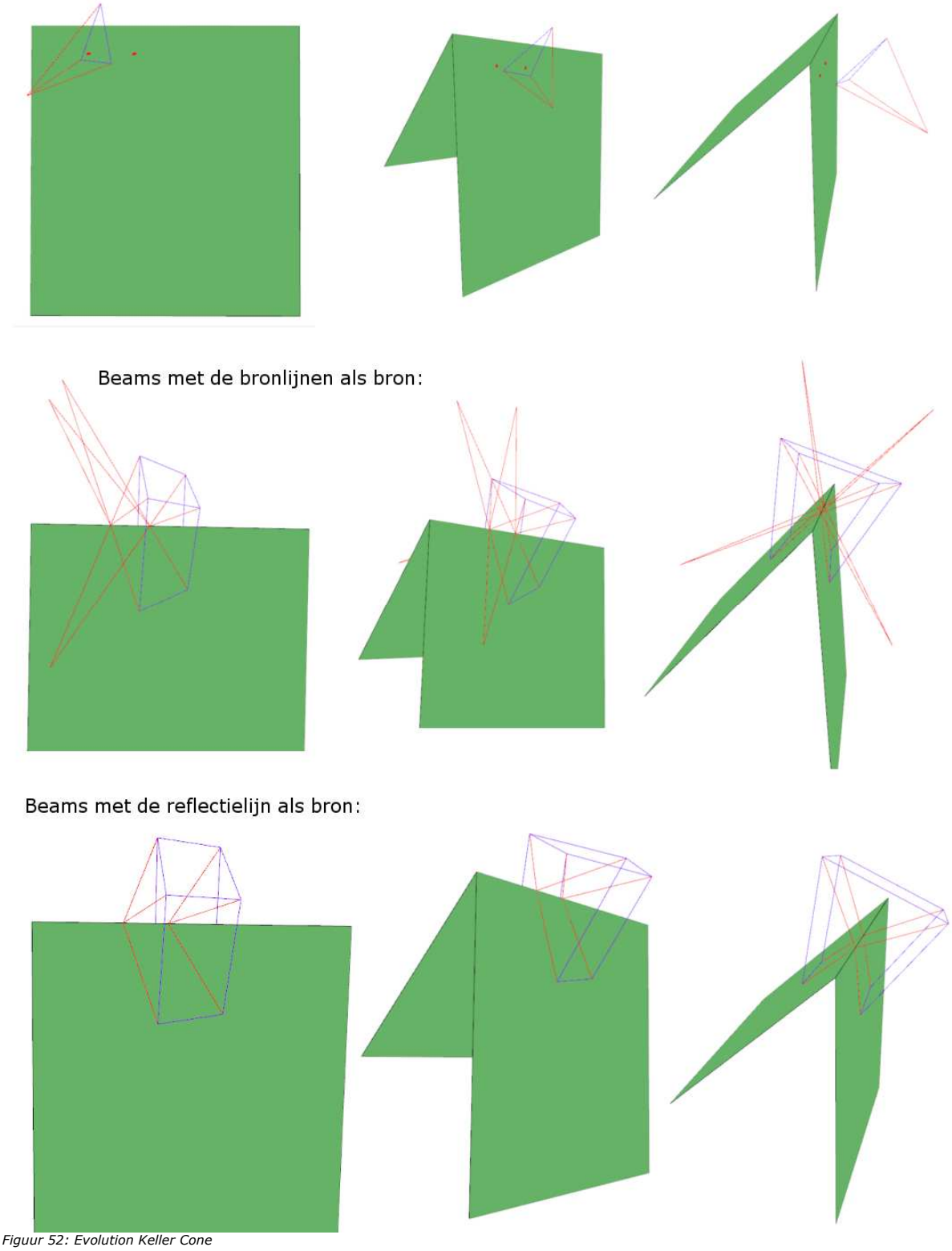


Figuur 51: Keller Cone met (overdreven) deviatie

- **De reflectielijn als bron**

In het verdere verloop van de scriptie zullen de rays van de Keller Cones worden getekend vanuit de reflectielijn en dus niet meer vanuit hun oorspronkelijke bronlijnen.

Indien de rays niet uit de reflectielijn worden verstuurd, zullen de rays botsen met de grenslijn waarop de Keller Cone staat. Voor deze reden worden de rays pas verstuurd vanaf de grenslijn.



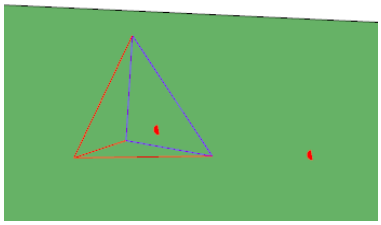
Figuur 52: Evolution Keller Cone

6) Creeping waves, een specifieke diffractie

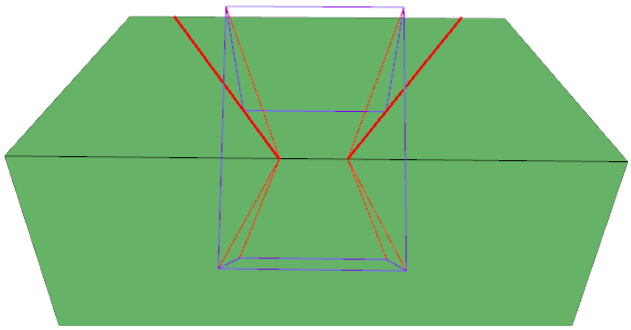
Creeping waves planten zich voort op het buur element in de schaduwzone van de invallende golf. Indien het buur element niet in de schaduwzone ligt, bestaat er geen creeping wave.
Een creeping wave heeft als bron een deel van de grenslijn van een element, en zal ook botsen op een andere grenslijn van hetzelfde element. Door de botsing zal opnieuw een Keller Cone worden gemaakt en eventueel ook een creeping wave als er een schaduwzone is. De grenslijn waarop de creeping wave botst kan ook gedeeld zijn met meerdere elementen, zodat de correcte hoek tussen de elementen moet berekend worden. Een herhaling van diffracties op een creeping wave zorgt bijvoorbeeld voor het effect dat een geluidsgolf rond een cilinder buigt (dit werd reeds besproken in hoofdstuk *9.a.v Creeping waves, een specifieke diffractie*) .

Een creeping wave wordt gesimuleerd als een tweedimensionale beam die over een element kruipt. Ze zijn het gevolg van diffractie, en dus komen ze samen voor met een Keller Cone.

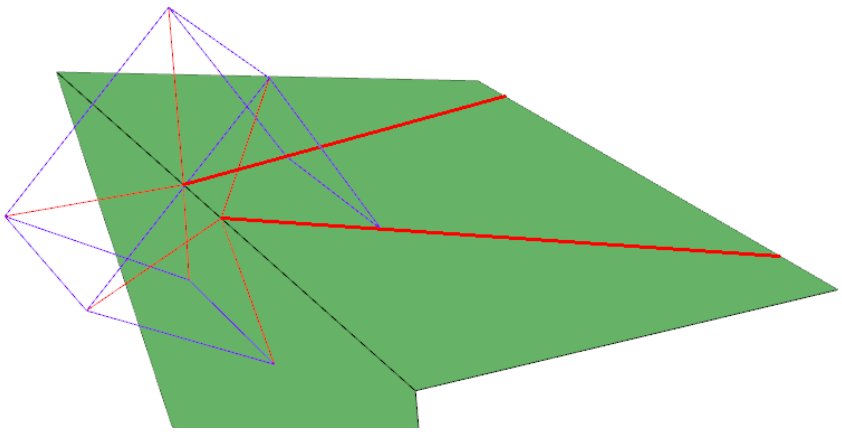
Bij een standaard Keller Cone worden de twee rays die het dichtst bij een element - en in de schaduwzone liggen gekozen als tweedimensionale creeping beam.
Figuur 53 toont een invallende beam die een schaduwzone heeft met het buur element. De invallende beam maakt een standaard Keller Cone op de grenslijn van het intersectie- en het buur element. Deze Keller Cone zal ook een creeping beam maken in de schaduwzone van de invallende beam. De creeping beam botst op zijn beurt met een andere grenslijn van het buur element.



Figuur 53: Creeping beam, invallende beam



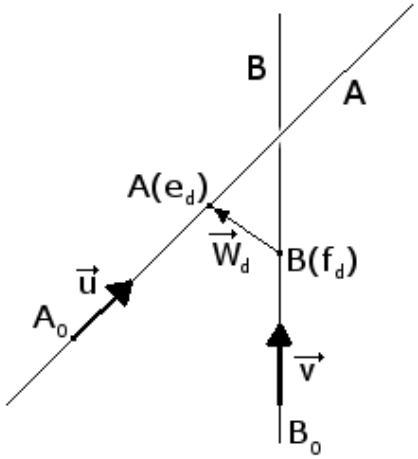
Figuur 54: Creeping beam, vooraanzicht



Figuur 55: Creeping beam, zijaanzicht

Er werd reeds vermeld dat de rays die het dichtst bij een element liggen een kleine deviatie hebben. Door deze deviatie zullen de creeping beams niet exact met een andere scheidingslijn botsen. De oplossing om een scheidingslijn te detecteren met een gedevieerde ray is de kortste afstand tussen de ray en elke scheidingslijn te berekenen. De gebruikte methode om de kortste afstand tussen twee lijnstukken te berekenen is die van Eberly (2000), genaamd calculus. Het is een afgeleide methode van Teller (Bibliografie [19]), de Teller methode gebruikt cross products en intermediaire vlakken waardoor deze methode trager is dan die van Eberly, daarom is er gekozen om Eberly te implementeren.

Twee lijnstukken A en B, die niet parallel staan ten opzichte van elkaar, zijn het dichtst op de unieke punten $A(e_d)$ en $B(f_d)$. De rechte die deze punten verbindt staat loodrecht op beide lijnstukken, geen andere rechte tussen de twee lijnstukken heeft deze eigenschap. Dus de projectie van de rechte op de lijnstukken is twee keer gelijk aan 0.
Anders geschreven geeft dit: $u \cdot W_d = 0$ en $v \cdot W_d = 0$.
Ook is gekend dat $W_d = A(e_d) - B(f_d) = W_0 + e_d \cdot u - f_d \cdot v$ (met $W_0 = A_0 - B_0$).



Figuur 56: Kortste afstand tussen lijnstukken

In Figuur 56:
W_d = de vector die de twee unieke punten verbindt.
u = de eenheidsvector op lijnstuk A
v = de eenheidsvector op lijnstuk B
A_0 = een willekeurig punt op lijnstuk A
B_0 = een willekeurig punt op lijnstuk B

De combinatie van beide geeft twee lineaire vergelijkingen:
$(u \cdot u)e_d - (u \cdot v)f_d = -u \cdot W_0$
$(v \cdot u)e_d - (v \cdot v)f_d = -v \cdot W_0$

e_d en f_d wordt uit deze vergelijk afgeleid:
$e_d = [n * q - o * p] / [m * o - n * n]$
$f_d = [m * q - n * p] / [m * o - n * n]$

Met
$m = (u \cdot u)$
$n = (u \cdot v)$
$o = (v \cdot v)$
$p = (u \cdot w_0)$
$q = (v \cdot w_0)$

De afstand tussen de punten e_d en f_d bepaalt de kortste afstand tussen de twee lijnstukken.
Deze methode wordt voor e_d en f_d herhaalt voor alle grenslijnen van het element waarop de creeping beam zich bevindt.

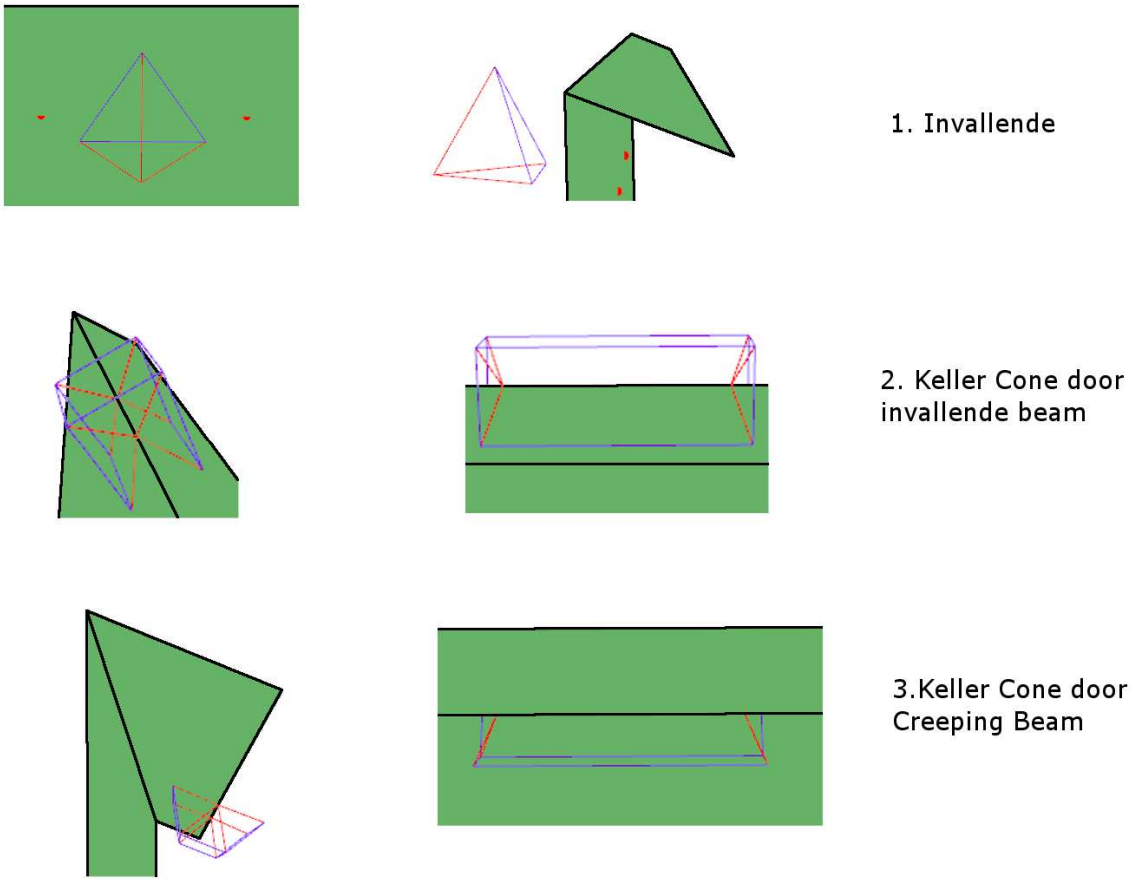
Op het eerste zicht lijkt deze methode complex om simpelweg de snijpunten van twee lijnstukken te zoeken. In principe is het exacte snijpunt van twee lijnstukken makkelijk te vinden met beide lijnvergelijkingen aan elkaar gelijk te stellen, maar dit kan enkel gebruikt worden als de lijnstukken exact op elkaar botsen (in hetzelfde ruimtelijk vlak liggen).

Stel dat de creeping beam geen deviatie heeft en zich op of in het element verplaatst. Theoretisch zal de creeping beam dan met een grenslijn van het element botsen. In de praktijk is het heel moeilijk om twee lijnstukken exact te laten botsen omdat er steeds een kleine afrondingsfout zal gebeuren, zeker op grote modellen is dit zo goed als onmogelijk.

De snijpunten van de creeping beam op een grenslijn zijn gekend. De volgende stap is controleren hoe de nieuwe Keller Cone gelegen is en of er al dan niet een creeping beam ontstaat.

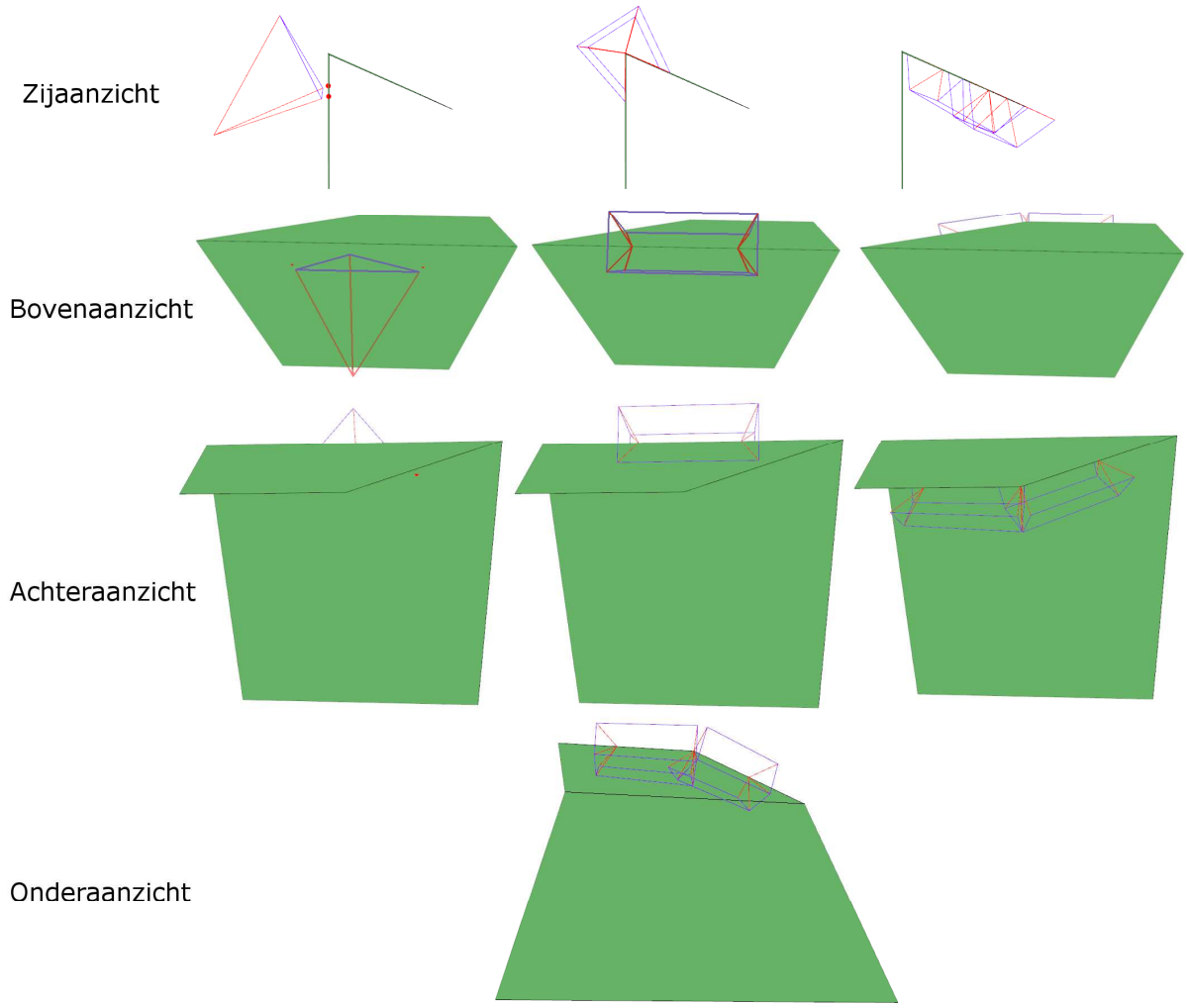
Diffractie van een creeping beam maakt een nieuwe Keller Cone op een gedeelte van de scheidingslijn. De begrenzing van de nieuwe Keller Cone is afhankelijk van het wel of niet bestaan van een schaduwzone. Indien er een schaduwzone bestaat, is de nieuwe Keller Cone begrensd door de schaduwzone van de invallende golf en het buur element. Een creeping beam ontstaat in de schaduwzone op het buurobject. Indien er geen schaduwzone bestaat is de nieuwe bron begrensd door het intersectie- en buur element. Omdat er geen schaduwzone bestaat, is hier geen creeping beam op het buur element.

Bijvoorbeeld een creeping beam op een element dat geen aangrenzende buren heeft (een edge) wordt er een 180° Keller Cone gemaakt:



Figuur 57: Keller Cone door een Creeping Beam

Een creeping beam kan, net zoals een vrije of een astigmatische beam, ook één of meerdere hoekpunt(en) van een element bevatten, deze creeping beam wordt dan ook opgesplitst. Bij vrije beams worden de normalen van de beam gecontroleerd, maar omdat creeping beams twee dimensionaal zijn, is deze methode hier niet bruikbaar.
Tijdens het zoeken naar de twee intersectie punten van een creeping beam wordt er numeriek bijgehouden met welke grenslijn van het element een ray van de creeping beam botst.



Figuur 58: Creeping beam op hoek van element

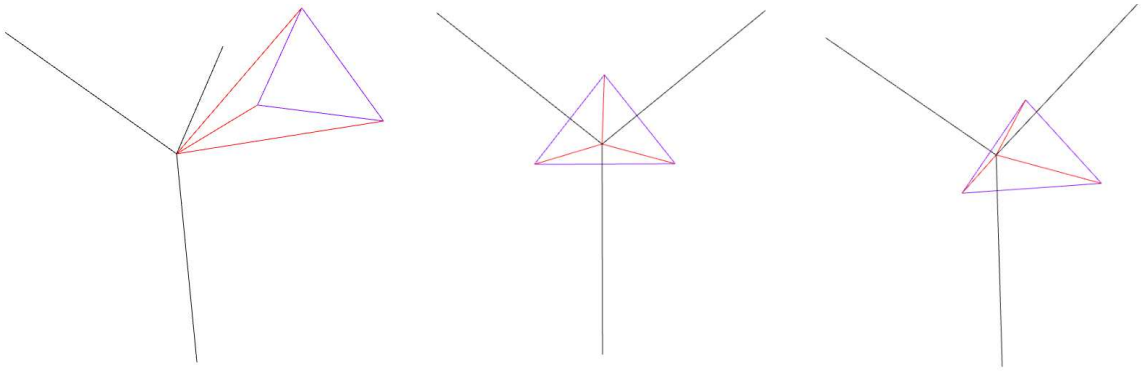
Stel dat de grenslijnen van het intersectie element in wijzerzin zijn genummerd, in bovenstaand voorbeeld botst de creeping beam met grenslijn nummer één en twee. Zo weten we hoeveel hoeken er tussen de snijpunten zijn (2 – 1) en als de hoekpunten ook in wijzerzin genummerd zijn weten we ook welk hoekpunt van het element zich in de creeping beam bevindt.
Net zoals bij de vrije beam zal er dan een Keller Cone gemaakt worden van een intersectiepunt naar het hoekpunt en een Keller Cone van het hoekpunt naar het ander intersectiepunt.

v. Controle fieldpoint in beam

Een fieldpoint is een punt in de virtuele wereld dat dient om geluidsstralen op te vangen, zoals een microfoon. Net zoals geluidsbronnen is het ook mogelijk dat er meerdere fieldpoints in een virtueel model bestaan.

Het doel van een Geometrical Acoustics techniek is het pad van een geluidsbron naar een fieldpoint te zoeken. Elke keer een beam verzonden wordt (geluidsbron, reflectie of diffractie) moet gecontroleerd worden of er een fieldpoint binnen een beam ligt.

Deze controle gebeurt weerom aan de hand van enkele logische vector bewerkingen.
Eerst worden de normalen van een beam gedefinieerd. Elke normaal behoort tot een dragend vlak van de beam, een dragend vlak wordt begrensd door 2 rays. Een normaal van een dragend vlak staat loodrecht op het respectievelijke vlak met als richting weg van het centrum van de beam.

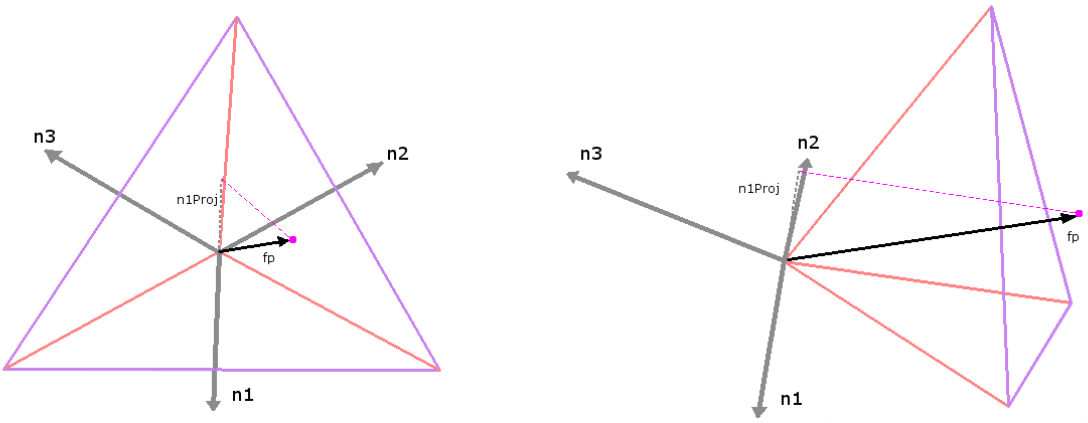


Figuur 59: Normalen van een driehoekige beam

De normaal van een dragend vlak wordt berekend door het cross product van de rays behorende tot dat vlak. Na het cross product kan de normaal gericht zijn naar of weg van het centrum van de beam, de richting moet dus gecontroleerd worden. Hiervoor wordt de normaal geprojecteerd op de ray van de beam die niet behoort tot het dragend vlak van de berekende normaal. Indien deze berekening positief is moet de normaal worden omgedraaid (vermenigvuldigen met -1).
Figuur 59 toont de normalen van een driehoekige beam. Zoals te zien zijn de normalen gericht weg van het centrum van de beam.
Een vierhoekige beam heeft dan vier normalen, deze worden op dezelfde methode berekend als hierboven.

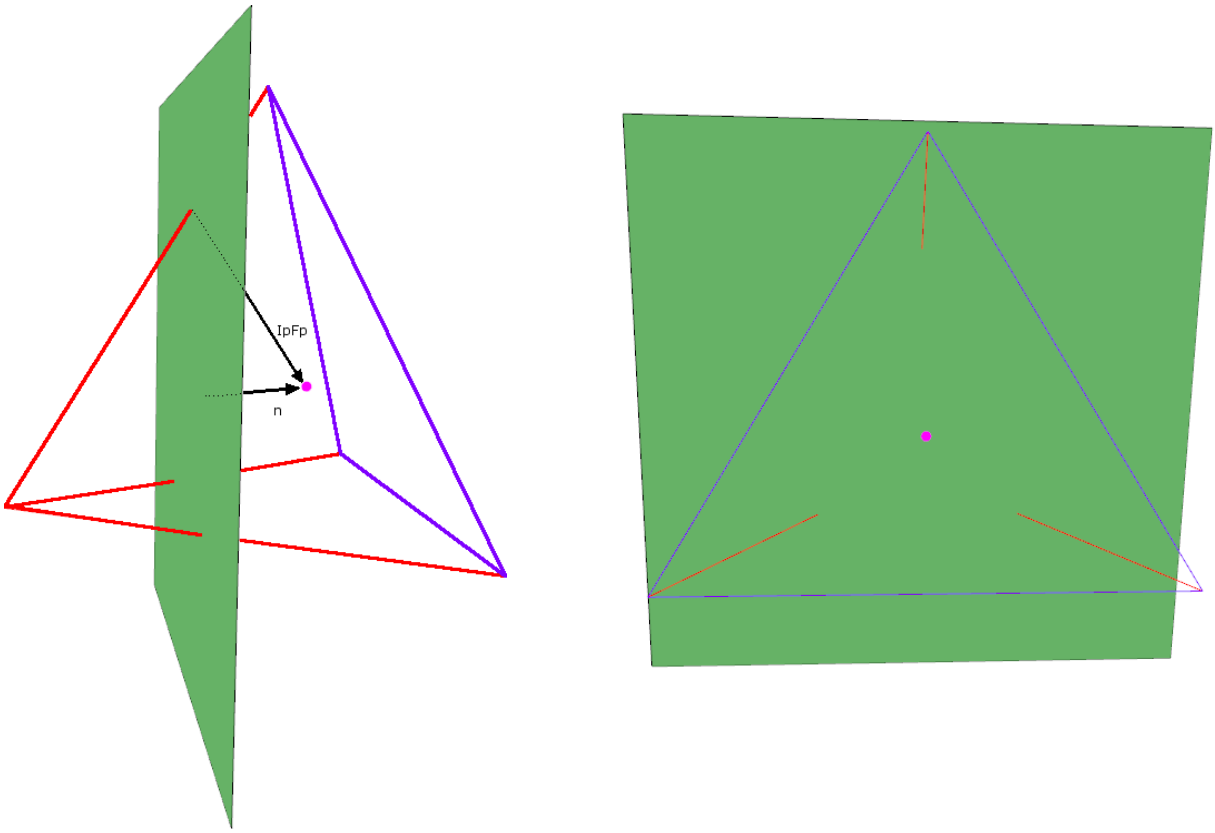
Met de normalen gekend kan, nu gecontroleerd worden of het fieldpoint in de beam ligt. Hiervoor wordt een vector (fp) gedefinieerd met als bron het bron punt van een ray en als richting het fieldpoint. Deze vector wordt geprojecteerd op elke normaal van de dragende vlakken van de beam. Indien deze lengtes van de geprojecteerde vectoren negatief zijn, bevindt het fieldpoint zich binnen de grenzen van de beam.

Op Figuur 60 is de projectie van vector fp op normaal n1 getekend, deze heeft duidelijk een negatieve lengte ten opzichte van n1.



Figuur 60: fieldpoint in beam

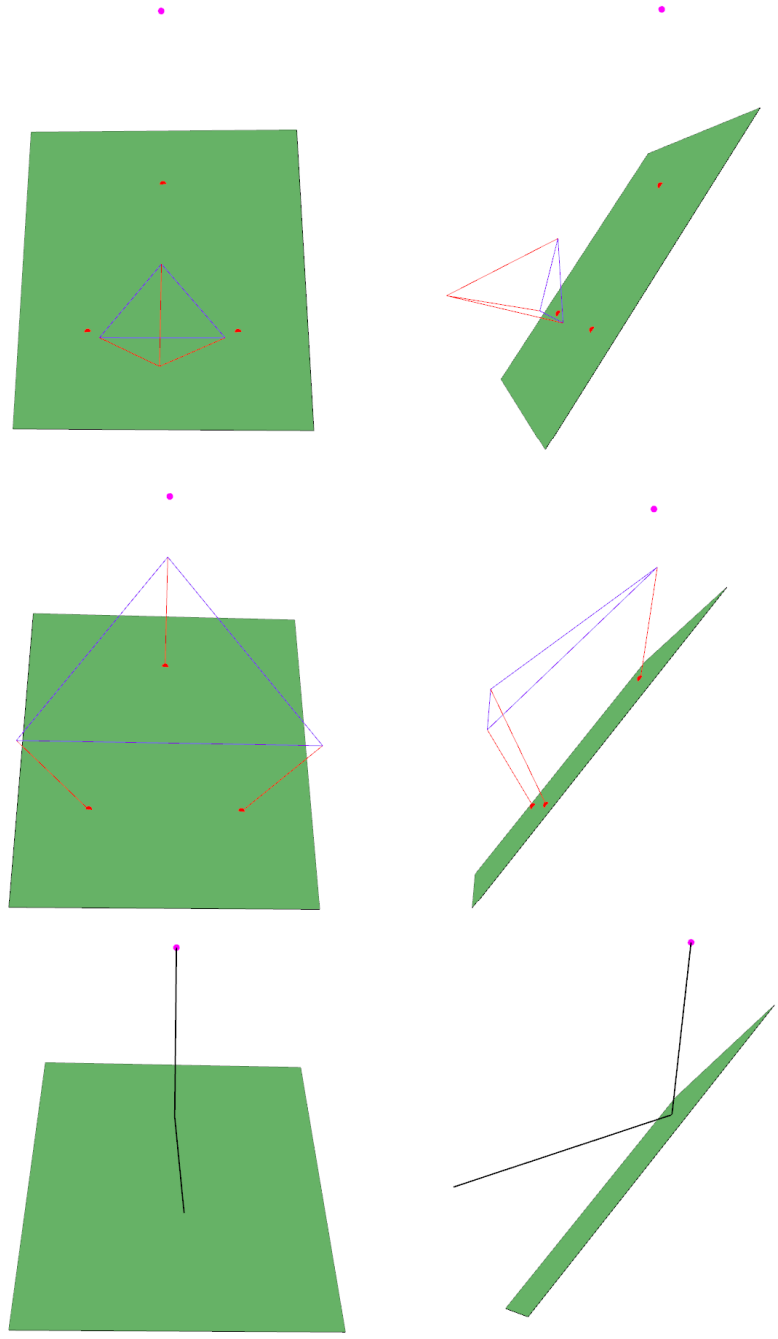
De besproken methode hierboven houdt geen rekening met de afstand van de rays. Zo is het mogelijk dat een fieldpoint gevonden wordt die achter een element ligt, wat dus niet correct is.
Daarom moet er gecontroleerd worden of het fieldpoint voor het botsingselement ligt.



Figuur 61: Fieldpoint achter element

Deze controle vraagt opnieuw de projectie van een vector op een andere vector.
De eerste vector (IpFp) heeft als bron het intersectie punt van de ray en als richting het fieldpoint.
De tweede vector (n) is de normaal van het element, in de richting van het fieldpoint.
De lengte van de projectie van de eerste - op de tweede vector moet negatief zijn. Indien de projectie positief is staat het fieldpoint achter een element (zoals op *Figuur 61: Fieldpoint achter element*).

Indien het fieldpoint in de beam en niet achter een element ligt, moet de weg van het fieldpoint naar de geluidsbron worden gevonden. Dit gebeurt door middel van backtracking.
Vanuit het fieldpoint wordt een weg gezocht naar het vorige reflectie element.
Elke keer een beam botst op een element wordt er bijgehouden op welk element de beam botst en het spiegelpunt waaruit zijn dragende rays worden berekend. Met deze informatie kan de weg van fieldpoint naar bron worden berekend.



Figuur 62: Voorbeeld opbouw geluidspad

Een voorbeeld waarbij een beam een fieldpoint na reflectie vindt, is te zien in Figuur 62.
De weg van het fieldpoint naar de bron bestaat uit twee wegen.

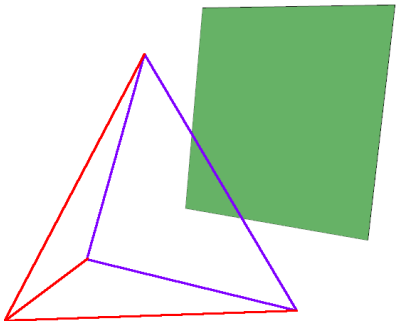
Het eerste deel:
Eerst wordt er een lijn gedefinieerd van het fieldpoint naar het spiegelpunt achter het element. Van deze lijn en het element wordt het intersectie punt berekend. Dit intersectie punt zal de bron zijn van het tweede deel van het geluidspad.

Het tweede deel:
Het tweede deel is eenvoudig een lijn van het intersectie punt (uit het eerste deel) tot de geluidsbron.
Deze delen samen geplaatst geeft het geluidspad.

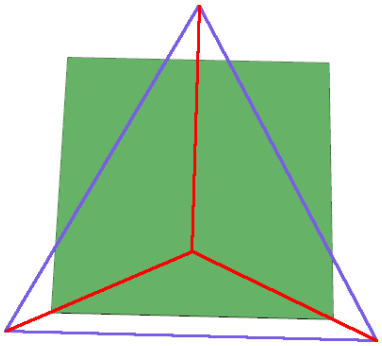
1) Uitzonderingen

- **Beams die over een element bewegen**

Het kan voorkomen dat beam een fieldpoint vindt die achter een element staat zonder diffractie. Dit komt voor wanneer de beam groter is dan het element. De beam heeft geen enkele dragende ray die tegen het element botst en zal dus 'door' het element gaan. Bij de backtracing van de weg van het fieldpoint naar de geluidsbron zal dan een deel van de weg door het element gaan.
Een zwevend element komt in realiteit zelden voor. Toch moest hier een beperking worden opgelegd. Bij de backtracing van elk deel van de weg, wordt er gecontroleerd of het deel niet door een element gaat. Indien het wel door een element gaat vervalt de volledige weg.



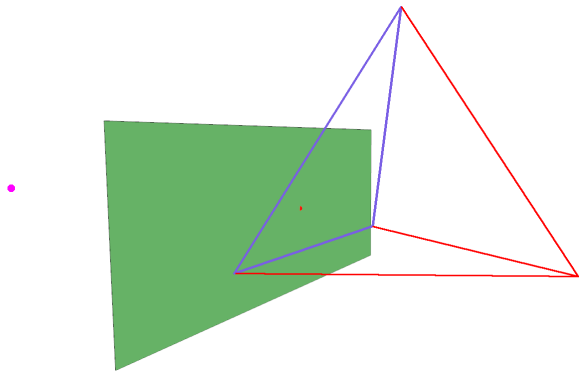
Figuur 64: Beam door element, zijaanzicht



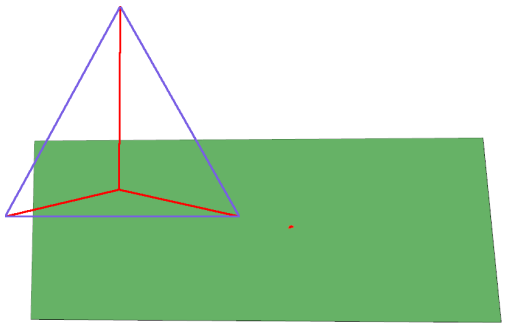
Figuur 63: Beam door element, vooraanzicht

- **Beams die niet volledig op een element botsen**

Beams waarvan de rays niet volledig op een element botsen worden niet gebruikt om fieldpoints te zoeken. Zoals te zien in Figuur 65, bevindt één fieldpoint zich in de beam maar is toch niet bereikbaar met een rechte lijn van de bron van de beam zonder door het element te gaan. Uiteraard bestaat er wel een geluidspad van bron tot fieldpoint door diffractie.



Figuur 65: Fieldpoint achter element, zijaanzicht



Figuur 66: Fieldpoint achter element, vooraanzicht

Stel nu dat het fieldpoint uit Figuur 65 een beetje hoger staat en dat het wel direct bereikbaar is vanuit de bron, zonder door het element te gaan. Dit geluidspad wordt dan genegeerd omdat de beam niet volledig tot in de oneindigheid door loopt. Uiteraard is dit te verhelpen door kleinere beams te gebruiken of door de beam te laten opsplitsen op de grenslijnen. De kans dat een bestaand geluidspad wordt genegeerd is hier dus minimaal.

d. Besluit

In vergelijking met het Triangular Beam Tracing algoritme bevat het Adaptive Beam Tracing algoritme:

- meer akoestische effecten, dankzij de diffractie en creeping waves,
- en springt het algoritme nauwkeuriger om met beams, dankzij de adaptieve opsplitsingen.

Binnen Raynoise is de overgang van TBT naar ABT daarom dan ook van groot belang.

Een nadeel van het ABT algoritme is dan weer dat er enorm veel nieuwe beams worden aangemaakt wat veel geheugen vraagt. Het extra geheugengebruik speelt in tegen het belangrijkste voordeel van Geometrical Acoustics. Het voordeel van een GA methode ten opzichte van Numerical Computational Method is het minder geheugengebruik. Een strikte beperking op het opsplitsen van beams is dan ook zeker nodig.

11. Implementatie van de opdracht

Er zijn in deze scriptie weinig codevoorbeelden te zien omwille van het feit dat er met LMS een privacy agreement werd opgesteld, dat mij verplicht om specifieke codevoorbeelden niet weer te geven. De methodes worden echter wel besproken.

De originele versie van Raynoise gebruikt het Traingular Beam Tracing algoritme. Wegens nieuwe, verbeterde, technieken wil LMS International één van deze verbeterde technieken gebruiken in Raynoise. De gekozen techniek is het Adaptive Beam Tracing algoritme.

Raynoise bestaat uit verschillende lagen: de virtualisatie, de solver, een methode die materiaal eigenschappen toepast, een methode die de resultaten van de solver verwerkt, een methode die intensiteit waarden afbeeldt in functie van de tijd, … .
De solver is het geometrical acoustics algoritme dat zoekt naar paden tussen fieldpoints en geluidsbronnen. In hoofdstuk 11.a wordt besproken hoe de methode wordt vervangen door het Adaptive Beam Tracing algoritme.

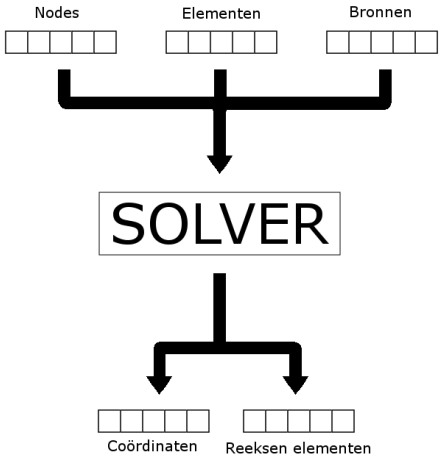
De nieuwe solver behandelt de akoestische eigenschappen in verschillende stappen, deze worden besproken in hoofdstuk *11.b* .

Er werd reeds aangehaald dat het geheugengebruik in een Adaptive Beam Tracing algoritme strikt gelimiteerd moet worden. In hoofdstuk *11.c* worden enkele manieren besproken om het aantal nieuwe beams te limiteren.

Verder worden in hoofdstuk *11.d* enkele technieken besproken die de performantie verbeteren.

a. Integratie in Raynoise

Het nieuw algoritme kan eenvoudig in Raynoise ingevoegd worden. De input van de nieuwe solver zijn enkele arrays die de virtuele objecten -, de geluidsbronnen – en de fieldpoints definiëren. De output van de nieuwe solver is een array die de paden tussen geluidsbronnen en fieldpoints definieert.



Figuur 67: Schema input / output solver

i. Input

Een **virtueel model** wordt als volgt gedefinieerd: een eerste lijst bepaalt de nodes als x-, y-, z-coördinaten. Een tweede lijst definieert de elementen door middel van de nodes; het type van het element en de bepalende nodes. Er zijn drie types elementen: een vierhoek (type vier), een driehoek (type drie) en een fieldpoint (type één).
Een fieldpoint wordt dus samen met het model gedefinieerd als een element.

De **geluidsbron** wordt gedefinieerd door zijn locatie en door het aantal opsplitsingen. Momenteel is het enkel mogelijk om een icosahedron als geluidsbron te kiezen. Later zullen er extra soorten geluidsbronnen worden toegevoegd.
Het is wel mogelijk om meerdere geluidsbronnen toe te voegen.

Verder zijn er nog enkele **argumenten** die gebruikt worden om het aantal beams en rays te begrenzen. Deze argumenten worden later besproken in hoofdstuk *11.c* .

ii. Output

Het doel van het Adaptive Beam Tracing algoritme is de geluidspaden van de geluidsbron naar een fieldpoint te zoeken. Later worden er formules toegepast op deze paden om de intensiteit van een geluidsbron ten opzichte van een fieldpoint te bepalen.

De output bestaat uit een reeks van geluidspaden.
Deze geluidspaden zijn gedefinieerd door de coördinaten waar - en de elementen waarop een geluidsstraal botst.

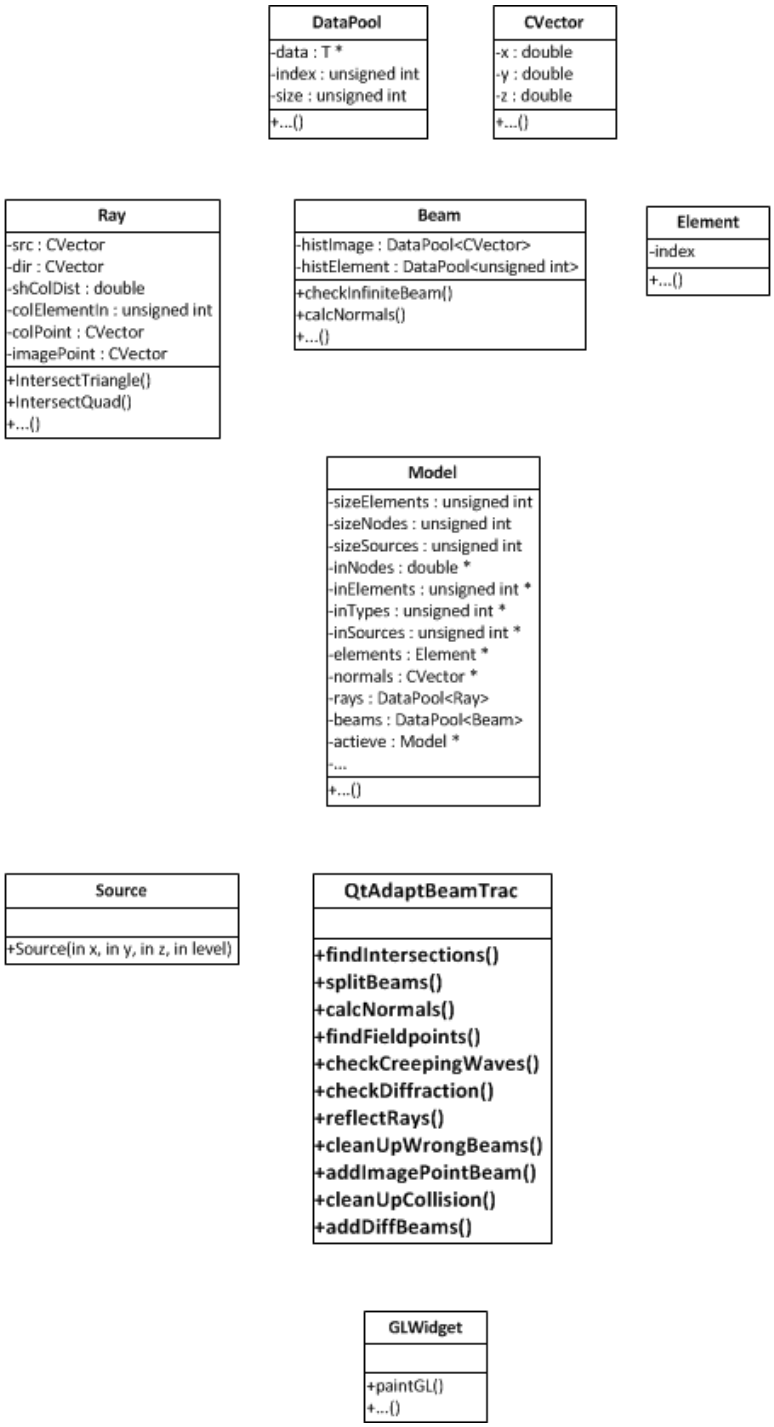
De reeks van **coördinaten** bepalen de plaatsen waar een geluidsstraal gereflecteerd of gediffracteerd werd. Hieruit kan dan de afstand van een geluidsstraal worden afgeleid.

Met behulp van de reeks van **elementen** kan dan de absorptie worden afgeleid.

b. Implementatie van de Adaptive Beam Tracing solver

i. Project architectuur

Voor de bespreking van de implementatie van de ABT solver is het belangrijk om een overzicht te hebben van de architectuur van het project. Hieronder staan de belangrijkste klassen vermeld.



Figuur 68: Overzicht van de belangrijkste klassen

De klasse "**Model**" bevat alle data die de solver nodig heeft (rays, beams, elementen, nodes, fieldpoints en tijdelijke waarden). De klasse bevat een statisch "Model" lid waarmee het mogelijk is om op een makkelijke manier van model te veranderen. Het data lid active staat gericht naar het actieve model, via deze pointer worden dan de data leden van de klasse aangevraagd. Op deze manier is het ook mogelijk om op een makkelijke manier van model te veranderen, het "Model" attribuut staat gericht naar het actieve model.
Het type 'DataPool' wordt later besproken.

```
const unsigned int sizeElements;
const unsigned int sizeNodes;
const unsigned int sizeSources;

double * inNodes;
unsigned int  * inElements;
unsigned int * inTypes;
unsigned int * inSources;

Element * elements;
CVector * normals;

unsigned int fpInd;

//RAYS AND BEAMS
DataPool<Ray> rays;
DataPool<Beam> beams;

//FIELDPOINT PATHS
unsigned int sizeFPRoutes;
unsigned int * lengthRoute;
double ** FPRoutes;
unsigned int ** elCol;

//TEMP DIFFRACTION BEAMS AND RAYS
DataPool<Beam> diffBeams;
DataPool<Ray> diffRays;

//CREEPING WAVES
DataPool<unsigned int> creapRays; //2 RAYS / CREEPING BEAM
DataPool<unsigned int> creapElemInd; //1 ELEMENT / CREEPING BEAM
DataPool<Beam> creapParent; //PARENT OF CREEPING BEAM
DataPool<CVector> creapDirNormIntElem; //DIRECTION NORMAL OF INTERSECTION ELEMENT
DataPool<CVector> creapOrigSource; //ORIGINAL SOURCE POINT OF 2 CREEPING WAVES

static Model * active;
```
Code VB 1: Data leden model.h

De klasse "**Ray**" definieert een ray. Een ray is gedefinieerd door een bron en een richting, beide zijn een x-, y-, z- coördinaat in het assenstelsel.
Een ray heeft ook volgende eigenschappen:

- De kortste botsingsafstand
- Het element waarop de ray is gebotst
- Het punt, binnen een element, waarop de ray is gebotst
- Het spiegelpunt waaruit de ray is berekend

De belangrijkste methodes van de ray klasse zijn:

- Een methode die controleert of de ray botst met een bepaald element (zie *Intersectie punten*).
- Een methode die de ray reflecteert, hier wordt bijvoorbeeld het botsingspunt de nieuwe bron.

De klasse "**Beam**" definieert een beam. Een beam is gedefinieerd door zijn (maximaal vier) dragende rays. Een beam heeft als functie de geluidspaden van geluidsbron naar fieldpoint te zoeken. Hiervoor is het nodig om de geschiedenis van de elementen waarop de beam botst te bewaren.

- De geschiedenis van spiegelpunten waaruit de rays van de beam zijn berekend
- De geschiedenis van elementen waarop een beam botst en heeft gebotst

De belangrijkste methode van de beam klasse zijn:

- Een methode die controleert of de beam volledig op een element botst, nodig voor het bepalen van diffractie.
- Een methode die de normalen van een beam berekent.

De klasse "**Element**" definieert een element. De klasse heeft als enig attribuut een index die verwijst naar de element tabel in de klasse Model. De methodes van de klasse zijn enkel getters and setters.

De klasse "**Source**" bevat één methode die een icosahedron opstelt met de nodige beams en rays.

De klasse "**QtAdaptBeamTrac**" bevat de methodes die alle akoestische eigenschappen definiëren, deze klasse wordt besproken in hoofdstuk *11.b.ii* .

ii. Akoestische eigenschappen in levels

De nieuwe solver test de akoestische eigenschappen in verschillende en specifieke stappen. De volgorde van deze stappen zijn belangrijk om de beams op een correcte manier te behandelen en te bewaren.

Beams worden level na level verwerkt. Een **level** betekent één akoestische eigenschap per beam, bijvoorbeeld één keer reflecteren op een element.

Een level bestaat uit een 13 tal stappen:

Figuur 69: Stappen per akoestisch level

Hierna volgt een gedetailleerde beschrijving, stap per stap, van de 13 te nemen stappen.

1) De geluidsbron

De eerste stap maakt een geluidsbron aan met eventueel verdere opsplitsingen, dit gebeurt in de klasse "Source".

Neem bijvoorbeeld *Code VB 2: source* : er worden twee arrays gedefinieerd (rays en beams). De array "rays" wordt vervolgens opgevuld met ray objecten waarvan de bron en richting coördinaten worden meegegeven. De array "beams" wordt opgevuld met een beam object dat gelinkt is met zijn dragende rays. Eventueel wordt de beam verder opgesplitst door de methode subBeam.

```
//DIRECTED BEAM
unsigned int sizeRays=3;
unsigned int sizeBeams=1;
DataPool<Ray> rays(sizeRays);
DataPool<Beam> beams(sizeBeams);
const float gr=(sqrt(5.0) + 1)/2; //Golden Ratio

rays[0]=Ray(CVector(x,y,z), CVector(0, gr, -1));
rays[1]=Ray(CVector(x,y,z), CVector(-1, 0.0, -gr));
rays[2]=Ray(CVector(x,y,z), CVector(1, 0.0, -gr));

rays.setIndex(3);
beams.setIndex(1);

beams[0]=Beam(1,    0   ,1, 2   ,0);

for(short int i = 1; i<=level; i++){
    QtAdaptBeamTrac::subBeams(i, rays, beams);
}
```

Code VB 2: source

Na opsplitsing worden de lokale arrays (rays en beams) toegevoegd aan de globale array in de klasse Model. De lokale arrays zijn van belang omdat het mogelijk is dat er verschillende geluidsbronnen bestaan. Elke geluidsbron moet dan afzonderlijk worden opgesplitst.

2) De intersectie punten

De tweede stap, findIntersections, zoekt naar de intersectie punten van de rays.
Deze methode werd eerder besproken in hoofdstuk *10.c.ii.1)* .

De besproken methode maakt enkel gebruik van driehoekige elementen, maar het virtueel model kan ook vierhoekige elementen bevatten. Om deze reden worden vierhoekige elementen tijdelijk opgesplitst in twee driehoekige elementen.

Indien een correct intersectie punt is gevonden worden de gegevens bewaard in het botsende ray object. De gegevens zijn: het intersectie element, het botsingspunt en de afstand van de ray tot het intersectie element.

3) Beams opsplitsen

De derde stap, splitBeams, splitst de beams op die niet volledig op één element botsen (zie hoofdstuk *10.c.iii De ABT sterkte: beams opsplitsen*).

4) Normalen van de beams berekenen

De vierde stap, calcNormals, berekent de normalen van de beams die in stap drie werden aangemaakt (zie *10.c.v Controle fieldpoint in beam*).

```
CVector * source = &histImage[getIndHist()-1];
CVector dirP0 = *source + *getRay(0).getDirection();
CVector dirP1 = *source + *getRay(1).getDirection();
CVector dirP2 = *source + *getRay(2).getDirection();

CVector P0minS = dirP0-*source;
CVector P1minS = dirP1-*source;
CVector P2minS = dirP2-*source;

normals[0] = P1minS.CrossProduct(&P2minS);
normals[1] = P2minS.CrossProduct(&P0minS);
normals[2] = P0minS.CrossProduct(&P1minS);

normals[0].Normalize();
normals[1].Normalize();
normals[2].Normalize();


//CHECK DIRECTION OF NORMALS
//calculate distance from vertex to opposite point in function of calculated normal
//if distance is negative, direction of normal is correct
//else multiply by -1
double dist;

for(int i = 0; i<maxRays; i++){
    CVector PMinP = (*getRay((i+1)%maxRays).getColPoint()) - (*getRay(i).getColPoint());

    dist = PMinP.DotProduct(&normals[i]);

    if(dist<=0.000000001)
        normals[i]=normals[i]*-1;
}
```

Code VB 3: normalen berekenen voor driehoekvormige beams

Uit *Code VB 3: normalen berekenen voor driehoekvormige beams* : eerst worden de dragende rays van de beam gedefinieerd als een vector (P0minS, …). Het crossProduct van elk aanliggend paar rays bepaalt dan een loodrechte vector op beide rays. Omdat er geen zekerheid is over de volgorde van de rays, moet de richting van de loodrechte vector gecontroleerd worden. Hier moet de normaal gericht zijn weg van het centrum van de beam.

5) Zoeken naar fieldpoints

De vijfde stap, findFieldpoints, onderzoekt of er fieldpoints zijn gevonden in beams (zie *10.c.v Controle fieldpoint in beam*). Indien een fieldpoint in een beam is gevonden, wordt het geluidspad van de geluidsbron naar het fieldpoint opgesteld.

Allereerst wordt er gecontroleerd of het fieldpoint in de beam ligt.

```
inBeam=true;
short int k=0;

while(k<maxInd && inBeam){
    CVector FpMinSk = Model::active->elements[j].getCVector(0) - (*Model::active->beams[i].getRay(k).getImagePoint());
    if(FpMinSk.DotProduct(&Model::active->beams[i].getNormal(k))>=0.0000001)
        inBeam=false;
    k++;
}
```

Code VB 4: zoek fieldpoint in beam

Vervolgens wordt er gecontroleerd of de beam volledig op één element botst en of het fieldpoint voor het element ligt.

```
//all rays collide with an element?
//else the beam doesnt collide at all (ignoring possible fieldpoint paths)
bool col=true;
short int l=0;
while(l<maxInd && col){
    if(Model::active->beams[i].getRay(l).getColElement()==0){
        col=false;
    }
    l++;
}

bool frontCol=true;
if(col){
    //check if fieldpoint lies behind or in front of the collision element

    //normal of collision element should be directed to source of beam
    CVector * elNormal = Model::active->beams[i].getRay(0).getColElement()->getNormal();

    //vector between collision point and imagePoint
    CVector dir = *(Model::active->beams[i].getRay(0).getColPoint()) -
        *(Model::active->beams[i].getHistImage(Model::active->beams[i].getIndHist() - 1));

    //project normal on dir
    //if negative, mirror the normal
    if(dir.DotProduct(elNormal)<0.00001){
        *elNormal=(*elNormal)*-1;
    }

    //vector between fieldpoint and collision point
    CVector FpMinSk = Model::active->elements[j].getCVector(0) - *Model::active->beams[i].getRay(0).getColPoint();

    if(FpMinSk.DotProduct(elNormal)>0.00001){
        frontCol=false;
    }
}
```

Code VB 5: Controle fieldpoint voor element

Als de vorige controles succesvol zijn verlopen wordt het geluidspad opgesteld zoals uitgelegd in hoofdstuk *10.c.v Controle fieldpoint in beam .*

6) Controle van de creeping waves

De zesde stap, checkCreepingWaves, controleert op welke grenslijn(en) een creeping wave botst. Op deze grenslijn komt dan een Keller Cone in de schaduwzone van de creeping wave en het buur element (zie *10.c.iv.6) Creeping waves, een specifieke diffractie*).

De beams en creeping waves die horen bij de nieuwe Keller Cone worden bewaard in tijdelijke tabellen en worden later toegevoegd bij de "echte" beams en creeping waves.
De reden hiervoor is omdat elk level maar één akoestische eigenschap heeft. Bijvoorbeeld de beweging van de creeping wave van zijn bron naar de grenslijn waarmee het botst is één akoestische eigenschap. Indien de creeping wave nog nieuwe beams zou aanmaken gebeuren er twee akoestische effecten.

7) Controleer beams op de diffractie eigenschap

De zevende stap, checkDiffraction, maakt een Keller Cone voor beams die niet volledig op één element botsen (zie *10.c.iv Diffractie: astigmatische beams en Keller Cones*).

Opnieuw worden de beams en creeping waves van de Keller Cone bewaard in tijdelijke tabellen, omwille van dezelfde reden als bij creeping waves.

8) De rays reflecteren

De achtste stap, reflectRays, reflecteert de rays waarvan, in de tweede stap, de intersectie punten waren gevonden.
Door deze stap zijn ook de beams, die volledig op één element botsen, gereflecteerd.

```
void QtAdaptBeamTrac::reflectRays(){
    //Reflect every ray that makes a collision with an element.
    //(reflect when collision distance is bigger then 0)
    for(int i = 0; i<Model::active->rays.getSize(); i++){
        if(Model::active->rays[i].getShColDist() >= 0.0001){
            Model::active->rays[i].reflect();
        }else{
            //ray goes in to infinity
        }
    }
};
```
Code VB 6: reflect rays

Het ray object zal zijn nieuwe bron aanpassen aan het intersectie punt en zijn nieuwe richting berekenen met behulp van het spiegelpunt.

```
void Ray::reflect(){
    CVector SminA = *getImagePoint() - *getColPoint();
    double perpDist = - SminA.DotProduct(getColElement()->getNormal()); //perpendicular distance

    //mirror current imagePoint over the collision element
    CVector newImagePoint = (*getColElement()->getNormal() * perpDist * 2.0) + *getImagePoint();

    this->setSource(*getColPoint());
    this->setImagePoint(newImagePoint);
    col=true;

    this->setDirection(*(getColPoint()) - newImagePoint);
};
```
Code VB 7: spiegelpunt berekenen

9) Foutieve beams verwijderen

De negende stap, cleanUpWrongBeams, verwijdert beams die
- niet op hetzelfde element botsen.
- helemaal niet botsen met een element, dus doorgaan tot in het oneindige.

10) Voeg spiegelpunt toe aan beam

De tiende stap, addImagePointBeam, voegt aan elke beam een spiegelpunt toe. Dit spiegelpunt wordt gebruikt om het geluidspad van fieldpoint naar geluidsbron op te maken.

Het spiegelpunt wordt in de eigenschappen van elke beam bewaard.

11) Normalen van de beams berekenen

Alle beams hebben vanaf stap tien een andere richting en bron punt. Daarom moeten opnieuw de normalen van de draagvlakken van elke beam worden berekend. Deze berekening gebeurt in de 11^{de} stap, calcNormals.

De methode is exact dezelfde als diegene gebruikt in stap vier .

12) Botsing eigenschappen opkuisen

Tijdens de tweede stap werden de intersectiepunten van de rays bewaard in het desbetreffende ray object. Deze punten zijn niet meer geldig na hun reflectie. Daarom moeten deze ook worden verwijderd, dit gebeurt in de 12^{de} stap, cleanUpCollision.

13) Beams uit de tijdelijke tabel verplaatsen naar de werkelijke tabel

Als laatste stap, addDiffBeams, worden de tijdelijke beams en creeping waves, die berekend werden in stappen zes en zeven, gekopieerd naar de echte tabellen.

Deze 13 stappen vormen één akoestisch level. Indien er meerdere akoestische levels gewenst zijn worden deze stappen herhaald, zonder opnieuw een source te maken.

c. Beam management

Dit deel bespreekt de limitatie van het aanmaken van nieuwe beams.

Naast de input en output objecten zijn er nog extra parameters waaraan de nieuwe solver moet voldoen. Deze parameters zijn van belang om het aantal beams en rays, dus het geheugengebruik, strikt te begrenzen. Sinds de nieuwe solver een adaptive algoritme is, worden er ook veel meer beams en rays aangemaakt ten opzichte van de oude solver.

Hieronder staat een lijst van de parameters. Deze worden toegelicht in het verdere verloop van dit hoofdstuk.

```
//**********HARD CODED BEAM PROPERTIES*****************************
maxDist=500.0; //THE MAXIMUM DISTANCE A RAY CAN TRAVEL
               //(a ray will not collide with an element farther then maxDist)
maxReflections=100; //THE MAXIMUM REFLECTIONS OF A BEAM
                    //(a beam can only reflect 'maxReflections' times)
maxDiffraction=2; //THE MAXIMUM DIFFRACTIONS OF A BEAM
                  //(a beam can only get diffrated 'maxDiffraction' times)

maxDivide=0; //THE NUMBER OF TIMES A FREE BEAM GETS DIVIDED IF THE BEAM COLLIDES WITH MORE THEN 1 ELEMENT

diffMinLength=0; //A KELLER CONE WILL NOT GET DIVIDED IF HIS REFLECTION LINE IS SMALLER THEN 'diffMinLength'
diffMaxSource=0; //THE NUMBER OF TIMES THE SOURCE LINE OF A KELLER CONE WILL GET DIVIDED
                 //(diffMaxSource=1; means 1 subdivision)
diffMaxRef=0; //THE NUMBER OF TIMES THE REFLECTION LINE OF A KELLER CONE WILL GET DIVIDED
              //(diffMaxRef=1; means 1 subdivision)

//DEVIATION FOR ASTIGMATIC BEAMS OF KELLER CONE
devCone = 0.008726644; //0.001745329rad = 0.1degree
                       //0.008726644rad = 0.5degree
//*****************************************************************
```

Code VB 8: Parameters voor beam management

De solver start met één of meerdere geluidsbronnen die elk hun aantal beams hebben. Naarmate een aantal levels worden toegepast, ontstaan er nieuwe beams. De nieuwe beams worden gemaakt door twee methodes:

- splitBeams: opsplitsing van een invallende beam
- checkCreepingWaves en checkDiffraction : het onderverdelen van een Keller Cone

Hieronder wordt er besproken hoe de methodes omgaan met begrenzingen van het aanmaken van nieuwe beams.

Verder wordt er nog besproken waarom het belangrijk is om oude beams te verwijderen.

i. Opsplitsen van een invallende beam

Een beam wordt opgesplitst als deze niet volledig op één element botst. De opsplitsing gaat door tot een maximale limiet. Deze limiet is bepaald door de parameter *maxDivide.*

Het kan gebeuren dat deze limiet te hoog is ten opzichte van de oppervlakte van de invallende beam. Veronderstel bijvoorbeeld twee vrije vlakken die naast elkaar staan. Een beam die op de grens van het eerst valt zal tot de limiet worden opgedeeld. Een nieuwe, kleine, beam zal hierna onmiddellijk weer op de grenslijn van het andere element botsen. Opnieuw wordt deze beam tot de limiet opgesplitst. Uiteindelijk worden er vele nieuwe beams gemaakt die even goed samen genomen kunnen worden.

De opsplitsing moet dus gelimiteerd worden, momenteel is dit nog niet het geval in de huidige solver. De eenvoudigste manier berekent de oppervlakte van de invallende beam aan de grenslijn en controleert of deze groter is dan een bepaalde constante. Hierbij wordt er gesuggereerd dat de oppervlakte evenredig is met de intensiteit van de beam, een kleine oppervlakte betekent dan dat de intensiteit van de beam verwaarloosbaar klein is.
Deze veronderstelling is zeker niet correct. Een zeer kleine beam kan makkelijk een grotere intensiteit hebben dan een grote beam. Bijvoorbeeld een kleine beam uit een geluidsbron ten opzichte van een grote beam, ook oorspronkelijk uit dezelfde geluidsbron, die enkele malen gereflecteerd werd.
Toch is dit een snelle, eenvoudige manier om het aantal beams te limiteren.

ii. Onderverdeling van een Keller Cone

Een standaard Keller Cone bevat drie astigmatische beams. Omdat deze beams soms te groot - en dus minder accuraat zijn kan de Keller Cone worden opgesplitst. Dit gebeurt door de oorspronkelijke bronlijnen op te delen en/of door de reflectielijn te verdelen. Het aantal opdelingen is bepaald door de parameters: *diffMaxSource* en *diffMaxRef*. *diffMaxSource* bepaalt het aantal opdelingen van de bronlijnen, *diffMaxRef* bepaalt het aantal opdelingen van de reflectielijn.

Opnieuw kan dit leiden tot een te groot aantal kleine beams.

De manier die momenteel geïmplementeerd is houdt rekening met de lengte van de reflectielijn. Als de lengte kleiner is dan een bepaalde lengte worden de bronlijn en de reflectielijn van de Keller Cone niet (verder) opgesplitst. De parameter diffMinLength bepaalt deze minimale lengte van een reflectielijn die nodig is om een Keller Cone verder op te splitsen.

De manier die besproken werd bij het opsplitsen van een invallende beam, waarbij er rekening wordt gehouden met de oppervlakte van de beam kan hier niet gebruikt worden. De reden hiervoor is dat er nog niet geweten is waar de astigmatische beams van de Keller Cone zullen botsen.

iii. Oude beams verwijderen

Oude beams zijn beams die al meerdere keren gereflecteerd zijn of gediffracteerd of beams die een redelijke afstand moeten afleggen voor ze botsen met een element. Zo'n beams hebben in werkelijkheid een verwaarloosbare intensiteit en kunnen dus verwijderd worden. De eerste parameter, maxDist, bepaalt de maximale afstand die een beam mag afleggen om te botsen met een element. Dit is dus niet de totale afstand die een beam heeft afgelegd.

De tweede en derde parameter, *maxReflections* en *maxDiffraction*, bepalen het aantal keer dat een beam een akoestisch effect mag ondergaan.

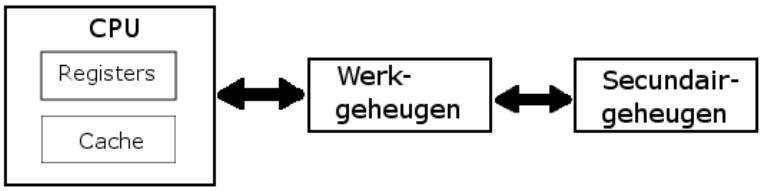
Opnieuw is deze methode weer een benadering met wat er in realiteit gebeurt. Om de methode accurater te maken is er meer informatie nodig over de elementen waarop een beam botst, zoals de absorptie coëfficiënt.

d. Performantie van de ABT solver

Al de data die het ABT algoritme nodig heeft, moet zo performant mogelijk verwerkt worden. **Data allocatie** is hier de key factor. Dit deel bespreekt snel de term "data allocatie" en hoe ermee wordt omgegaan in de ABT solver.

i. Wat is data allocatie?

Om de uitleg over data allocatie te verstaan is er een kleine kennis nodig van de processor architectuur. De processor is de centrale eenheid van een systeem die de berekeningen uitvoert.
De processor werkt rechtstreeks met zijn eigen cache geheugen of het werkgeheugen.
Het cache geheugen kan snel uitgelezen worden door de processor en wordt gebruikt door data die regelmatig nodig is. Het werkgeheugen (RAM geheugen) kan ook snel uitgelezen worden door de processor, het secundair geheugen (harde schijf) is meestal een mechanische data opslag en is de traagste van de drie opslag mogelijkheden.
Indien de processor iets nodig heeft van het secundair geheugen moet deze data gekopieerd worden naar het werkgeheugen.



Figuur 70: Eenvoudige voorstelling van de CPU architectuur

Letterlijk betekent data allocatie, het toewijzen van een geheugenplaats. Bij de declaratie van een variabele, van eender welk type, zal het programma een geheugenplaats reserveren in het werkgeheugen. Indien er geen plaats meer is in het werkgeheugen wordt er data verplaatst naar een secundaire dataopslag, deze regeling noemt **swapping** (paging).

Er bestaan twee manieren om geheugen te reserveren: statische - en dynamische allocatie.

1) Statische allocatie

Statische allocatie werd door de allereerste programmeertalen geïntroduceerd (vb: Fortran). De allocatie wordt in compiler-tijd uitgevoerd, dus vóór het eigenlijke programma is opgestart.
De compiler reserveert voor elke variabele een bepaald gedeelte van het geheugen. Deze grootte kan niet meer wijzigen tijdens de uitvoering van het programma.
Het gereserveerde geheugen kan ook niet meer worden vrijgegeven tijdens de uitvoering van het programma. Het is dus belangrijk om op voorhand te weten hoeveel geheugen een variabele zal nodig hebben.

Bijvoorbeeld de allocatie van een tabel met vijf integers:

```
int array[] = {5, 4, 3, 2, 1};
```

De compiler zal, in het vrije geheugen, een data blok reserveren waarin vijf integers kunnen bewaard worden. array[0] geeft dan de integer terug op de eerste plaats van het data blok.

Een voordeel van statische allocatie is dat alle data achter elkaar wordt bewaard in het geheugen. Wetende dat één integer 4bytes groot is, blijkt uit bovenstaand voorbeeld dat de waarde van array[1] exact 4bytes verder staan dan die van array[0].
Een belangrijk nadeel is dan weer dat er makkelijk buiten het gereserveerde data blok kan worden geschreven. Wat grote gevolgen kan hebben, aangezien het dan mogelijk is dat een andere variabele een andere waarde krijgt. Zo zal array[11] een onbekende waarde voorstellen, omdat de array maar tot index 4 is gedeclareerd.

2) Dynamische allocatie

Dynamische data allocatie werd na statische allocatie geïntroduceerd. Deze maakt het mogelijk om data allocatie uit te voeren tijdens de uitvoeringstijd van het programma.
In tegenstelling tot statische allocatie is het bij dynamische allocatie dus niet meer nodig om op voorhand te berekenen hoe groot een bepaalde array moet zijn.

Bijvoorbeeld de dynamische allocatie van een tabel:

```
int * array = new int [5];
```

De variabele, array, is nu gedeclareerd als een pointer die wijst naar een tabel van vijf integers.
De pointer kan makkelijk verplaatst worden naar een andere tabel. Bijvoorbeeld naar een tabel voor tien integers:

```
array = new int [10];
```

Door de introductie van dynamische allocatie zijn er ook nieuwe containers geïntroduceerd. Zo is er in C++ de sequentie container "list". Een list is een gelinkte lijst waarmee op een makkelijke manier gelinkte data kan worden bewaard.
Zo heeft de container een functie *push_back(type _in)* waarmee een bepaalde data wordt toegevoegd aan de gelinkte lijst.
De data van een gelinkte lijst staat niet vlak achter elkaar. In tegenstelling tot een normale tabel, staat de data van een gelinkte lijst verspreid over het geheugen. De functie *push_back(type _in)* roept de *allocate()* functie op waarmee een plaats in het geheugen wordt gezocht om de nieuwe data te bewaren.

In het voorbeeld hierboven werd een integer tabel herverdeeld van vijf integers naar tien integers, de pointer werd verplaatst naar een nieuwe locatie. Een belangrijk onderdeel werd nog niet vermeld, de initiële data (tabel van vijf integers) werd niet automatisch vrijgegeven na de nieuwe allocatie. Hiervoor is de term "delete" nodig. Indien er geen delete wordt uitgevoerd, blijft de initiële data in het geheugen en vormt het een **memory leak**. De correcte herverdeling gaat als volgt:

```
int * array = new int [5];
delete [] array; //verwijderen van een array ➔ []
array = new int [10];
```

3) Vergelijking van beide allocatie methodes

Het is duidelijk dat dynamische allocatie makkelijker te gebruiken is dan statische allocatie, toch zijn er nog enkele belangrijke aspecten die 'achter de schermen' gebeuren maar die een grote invloed kunnen hebben op de performantie en efficiëntie van een programma.

Het voordeel van dynamische allocatie is dat er dynamisch met **geheugen** wordt gewerkt. Het zal daarom dan ook, meestal, minder geheugen gebruiken dan statische allocatie.
Dynamische allocatie zal niet altijd minder geheugen gebruiken, de reden hiervoor is dat de gedefinieerde containers, zoals list, extra data gebruiken dan bij een gewone tabel.
Een list bestaat uit een reeks elementen die met elkaar verbonden zijn door pointers. Elk element zal dus ook een pointer moeten bevatten die verwijst naar het volgende element. Deze pointer heeft ook geheugenplaats nodig, op een 64bit systeem zijn dit 64bits.
Stel dat er een list wordt gedefinieerd van integers, een integer heeft 32bits geheugen nodig, de pointers zullen dan meer geheugen nodig hebben dan de eigenlijke data. Indien er veel data wordt bewaard en het geheugen beperkt is, is deze container absoluut geen efficiënte oplossing.

Een ander gevolg van dynamische data allocatie is de **extra overhead** ten opzichte van statische data allocatie. De extra overhead komt voor bij het toevoegen van nieuwe data en zelfs bij het overlopen van de data.

- **Data toevoegen**

Er werd reeds vermeld dat er een allocatie functie wordt opgeroepen bij een push_back oproep in de list container. Deze allocate bestaat uit volgende stappen:
- Windows memory allocatie wordt geblokkeerd
- Windows vraagt een data blok om één integer te bewaren
- De standaard integer constructor wordt opgeroepen
- Windows deblokkeert de memory allocatie
- De nieuwe data wordt gekopieerd naar het vrije data blok

Bij een statische array, die voldoende initieel geheugen over heeft, wordt er geen allocatie uitgevoerd. De nieuwe data wordt simpelweg gekopieerd naar een bepaalde index.
Indien de statische array geen vrij geheugen meer heeft moeten er data elementen uit de statische array worden verwijderd of moet het data element in een andere array worden bewaard. Data toevoegen bij een statische array bestaat dus niet altijd uit één operatie.

- **Data overlopen**

Bij het gebruik van dynamische containers zal data verspreid worden over het geheugen.
Indien er veel data wordt toegevoegd kan het voorkomen dat een deel van de data op secundaire dataopslag staat (swapping). Om de data te overlopen zal er dan ook data van het secundair dataopslag gekopieerd moet worden naar het werkgeheugen, wat niet erg efficiënt is.

Een statische array bewaart alle data vlak achter elkaar, de processor kan dan snel over alle data lopen. Tenzij de volledige statische array niet in het werkgeheugen past. Dan wordt een deel van de array bewaard op het secundair geheugen en moet er nog steeds geheugen worden gekopieerd om de array volledig te overlopen. Toch zullen er niet zoveel kopieeroperaties gebeuren als bij een dynamische container omdat één kopieeroperatie een grote reeks opeenvolgende data elementen bevat. Bij dynamische allocatie is er absoluut geen zekerheid dat alle overige data elementen in één kopieeroperatie worden gekopieerd.

ii. Data allocatie in de ABT solver

De solver maakt gebruik van beide allocatie types.
Statische allocatie wordt gebruikt voor data elementen die constant blijven tijdens het uitvoeren van de solver. Bijvoorbeeld de drie input arrays die het virtueel model definiëren, deze data arrays wijzigen niet.

Data elementen die wijzigen of worden verwijderd tijdens de werking van de solver, worden bewaard in een dynamische datastructuur. Bijvoorbeeld de rays en beams objecten, deze data zal na elk uitgevoerd level een andere waarde hebben.
De rays en beams objecten worden elk in een apart type datastructuur bewaard in de klasse "model.h" (zie *11.b.i Project architectuur*).
Deze twee groepen van objecten nemen het meeste geheugen in beslag, het is dus belangrijk om voldoende geheugen te alloceren. Maar ook weer niet te veel zodat het nog mogelijk is om andere data te bewaren (bijvoorbeeld de geluidspaden).

Er is dus een datastructuur nodig waar eenvoudig en efficiënt nieuwe elementen kunnen aan toegevoegd en van verwijderd worden. De makkelijkste oplossing om de dynamische data elementen te bewaren is in een dynamische container, die op een dynamische manier data in het geheugen bewaart. Maar door het extra geheugen en overhead van een dynamische container is dit niet de optimale oplossing. Het ABT algoritme heeft op zich al veel geheugen nodig om akoestiek te simuleren, hoe meer geheugen het algoritme kan gebruiken, hoe nauwkeuriger het resultaat zal zijn.

Deze datastructuur is geïmplementeerd in de klasse "DataPool.h".
Een DataPool is een bepaald type dynamische array, die zijn beschikbaar geheugen vergroot naarmate er data elementen worden aan toegevoegd. De DataPool klasse heeft volgende data leden:

```
template <typename T>
class DataPool {
private:
    T * data;
    unsigned int size;
    unsigned int index;

public:
    ...

};
```

Code VB 9: Data leden DataPool

Het data lid "data" is een pointer naar een bepaald type data element, dit is de dynamische array. Het data lid "size", bepaalt het maximaal aantal beschikbare plaatsen in het geheugen die gereserveerd werden. Het data lid "index", bepaalt de eerste vrije plaats van de array (één plaats verder dan het laatst toegevoegde data element).

Hieronder enkele functie leden van de DataPool klasse:

```
DataPool(unsigned int _size):size(_size), index(0){
    data = new T[_size];
};

void add(T& elem){
    if(index>=size){
        setSize(getSize()*2);
    }

    data[index]=elem;
    index++;
};

void setSize(unsigned int _size){
    if(_size > size){
        T * oldData = data;

        data = new T[_size];
        for(unsigned int i=0; i<getSize(); i++){
            data[i] = oldData[i];
        }

        delete [] oldData;

        size=_size;
    }
};

T &get(unsigned int _index)const {
    return data[_index];
};

void resetIndex(){
    index=0;
};
```

Code VB 10: Functie leden DataPool

De constructor van de klasse declareert een dynamische array van een bepaalde grootte. Bij het toevoegen van een nieuw data element wordt er gecontroleerd of de array nog beschikt over vrij geheugen, zoniet wordt de grootte van de array verdubbeld. Momenteel gebeurt de verdubbeling door simpelweg een nieuwe array aan te maken met dubbele grootte en de bestaande data elementen te kopiëren van de oude - naar de nieuwe array. Dit kan efficiënter, zie hoofdstuk *11.e Voorgestelde verbeteringen* .

e. Voorgestelde verbeteringen

Het doel van mijn opdracht was het ontwikkelen en implementeren van een Adaptive Beam Tracing solver en de belangrijkste akoestische eigenschappen, zoals bijvoorbeeld de creeping waves. Daarnaast was de performantie een – weliswaar - minder belangrijk aspect van de opdracht.

Op het vlak van performantie kan er dus nog wat worden verbeterd. Dit deel bespreekt enkele mogelijke verbeteringen.

i. De geschiedenis van een beam voorstellen in een boomstructuur

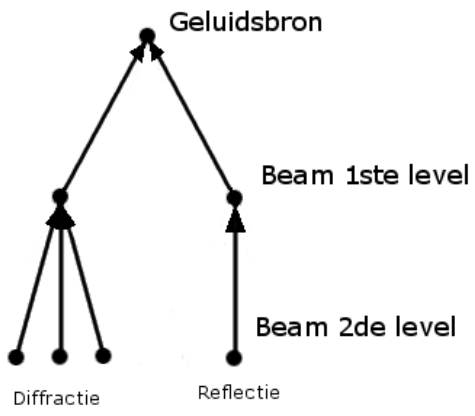
Om de geluidspaden op te stellen is het nodig de geschiedenis van elke beam te weten. Deze geschiedenis bestaat dan uit punten en elementen waarop de beam is gebotst.

Momenteel bewaart elk beam object zijn eigen geschiedenis, dit in een DataPool: histImage en histElement.

```
bool isTr; //A standard Beam has 3 rays, an astigmatic beam has 4 rays
unsigned int * rays;    //Refers to indexes in the global array of model.h

//TODO: Tree structure.
DataPool<CVector> histImage; //history of image points
DataPool<unsigned int> histElement; //history of elements the beam collided

unsigned int diffCtr;

CVector * normals;
bool nCalc; //normals calculated?
```

Deze manier bewaart redelijk wat dubbele informatie. Want de geschiedenis van een beam bestaat steeds uit de geschiedenis van zijn ouder beam (of de geluidsbron voor een beam uit het eerste level).
Neem bijvoorbeeld een beam die botst op de grens van een element, deze beam zal worden opgesplitst. De geschiedenis van de nieuwe, kleinere beams zal bestaan uit de geschiedenis van de invallende beam + de botsing op de grenslijn.

Een efficiënte manier om de geschiedenis van de beams te bewaren is door middel van een boomstructuur. De bovenste knoop stelt dan de geluidsbron voor, elk uitgevoerd akoestisch level maakt dan een nieuw niveau in de boomstructuur. Elke beam in de boomstructuur is gelinkt met zijn ouder, waardoor zijn geschiedenis kan worden gevormd.



Figuur 71: Beam geschiedenis

Figuur 71 toont hoe de boomstructuur in elkaar zit. De bovenste knoop is de geluidsbron, die twee beams verstuurt. Elk van beide beams verwijst naar de geluidsbron, indien één van beide beams een fieldpoint ontdekt kan het geluidspad worden opgemaakt.
Beide beams botsen op een element, de eerste beam wordt gediffracteerd en maakt drie nieuwe beams aan. De tweede beam wordt gereflecteerd, er wordt één nieuwe beam aangemaakt. Opnieuw wijzen de nieuwe beams naar hun ouder.

ii. Het virtueel model voorstellen in een boomstructuur

De methode waarmee wordt gezocht of een ray botst met een element is reeds besproken in hoofdstuk *11.b.ii.2) De intersectie punten*. Momenteel wordt elk element overlopen om te controleren of er een ray op botst. Dit is absoluut niet efficiënt en kan beter met een boomstructuur.

De boomstructuur zal het virtueel model opsplitsen in meerdere regio's. Wanneer er dan gezocht wordt achter het element waarop een ray botst, kunnen er regio's genegeerd worden. Bijvoorbeeld de regio's die in de negatieve richting van een ray liggen kunnen onmiddellijk genegeerd worden.

Er zal moeten onderzocht worden welke boomstructuur het best past bij dit project. Het is zeker niet de bedoeling dat de gekozen boomstructuur veel geheugen zal gebruiken, de nauwkeurigheid van de solver mag niet in gedrang komen.

De twee bekendste boomstructuren zijn: de drie dimensionele kd-tree en de octree.

iii. Efficiëntere allocatie van de DataPool

Indien een DataPool te klein is wordt de array gealloceerd naar een grotere DataPool. Deze allocatie is simpelweg een nieuwe array aanmaken, van een dubbele grootte, en de originele array pointer naar de nieuwe array laten verwijzen.

Dit kan efficiënter met behulp van de realloc functie van c++ .

*void *realloc(void *ptr, size_t size);*

De realloc functie verandert de geheugengrootte van het object, waar *ptr* naar wijst, naar een grootte die gedefinieerd is door de parameter *size*. Indien er genoeg geheugen is achter het object, wordt het object niet verplaatst en wordt er extra geheugen gereserveerd om te voldoen aan de gedefinieerde parameter *size*.
Indien er te weinig geheugen beschikbaar is achter het object, wordt het object nog steeds gekopieerd naar een andere locatie.

f. Debugging met QtOpenGL

Uiteindelijk zal de ABT solver geïntegreerd worden in een CATIA V5 omgeving. De CATIA V5 omgeving leek mij te omslachtig om de nieuwe solver in te testen. Want in het begin stadium van het project worden er nog geen grote en omslachtige modellen gebruikt. De modellen waarop getest moet worden zijn makkelijk zelf samen te stellen door de correcte node coördinaten in te geven. Daarom is er gekozen om de beams en modellen visueel te controleren met behulp van de Qt module QtOpenGL.

De QtOpenGL module bevat klassen die het mogelijk maken om drie dimensionale objecten te tekenen in een OpenGL omgeving. Bijvoorbeeld GL_POINTS, GL_LINES, GL_TRIANGLES en GL_QUADS.

De klasse in het project die de objecten weergeeft, "GLWidget.h", staat volledig onafhankelijk van het project. Het tekenen gebeurt enkel door de gepaste arrays in te lezen. Het is dus perfect mogelijk om later het Qt gedeelte te vervangen door de CATIA V5 omgeving.

Het resultaat van de solver werd enkel visueel gecontroleerd. Omdat er met kleine modellen werd gewerkt om de akoestische eigenschappen te integreren, was de visuele controle voldoende. In een later stadium, bij grote complexe modellen, zal de visuele controle niet meer mogelijk zijn. De resultaten van de solver zullen dan gecontroleerd moeten worden met een Nummerical Computational Method of door de implementatie van een testcase.

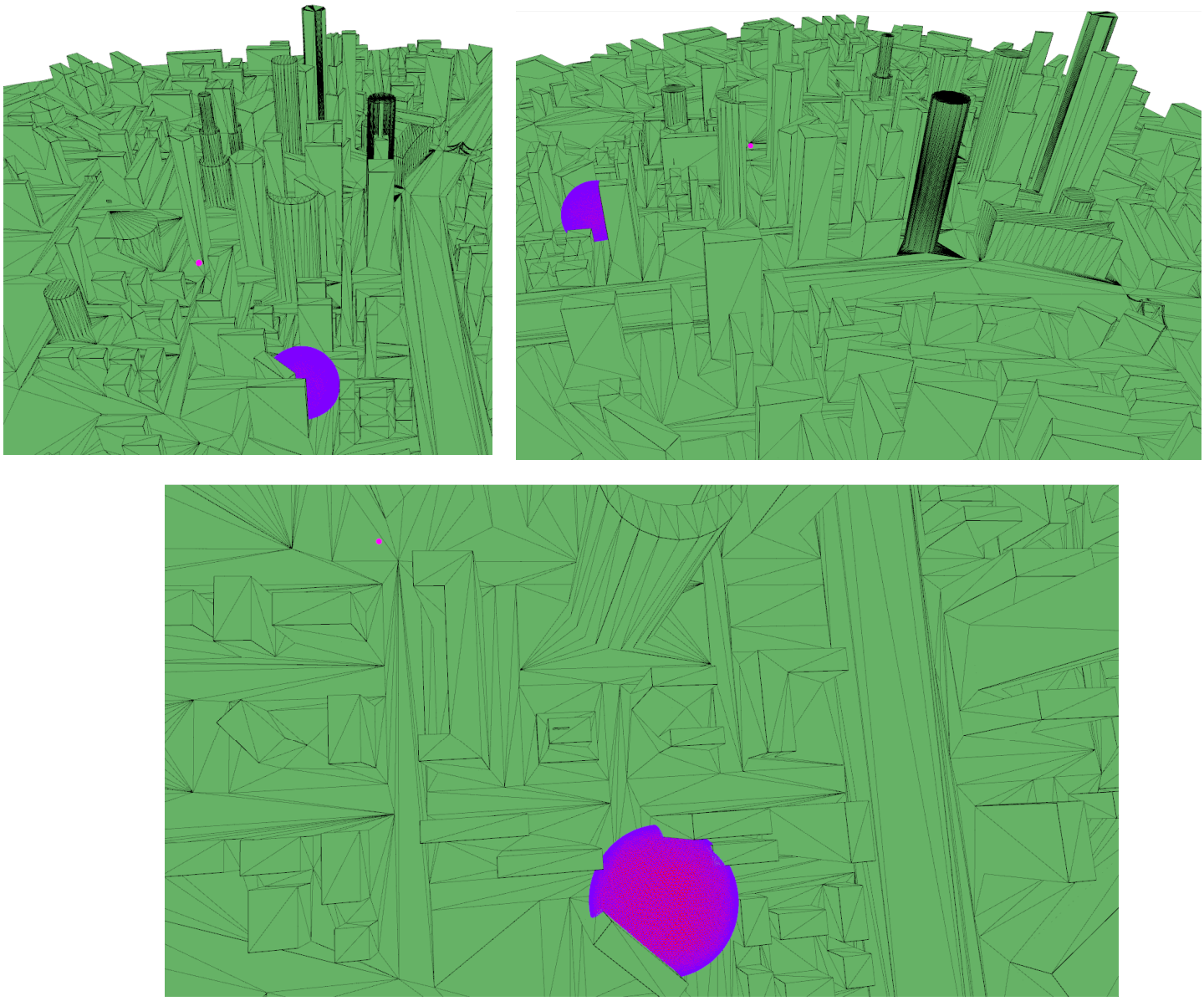
12. Voorbeelden

Nu volgt een voorbeeld waarin al de voorgaande akoestische eigenschappen worden toegepast in een redelijk complex model.

Het model is een virtuele stad met redelijk wat gebouwen met verschillende vormen en enkele sky scrapers. Als bron is een icosahedron gebruikt die vijf keer is opgedeeld, een fieldpoint staat enkele straten verder.

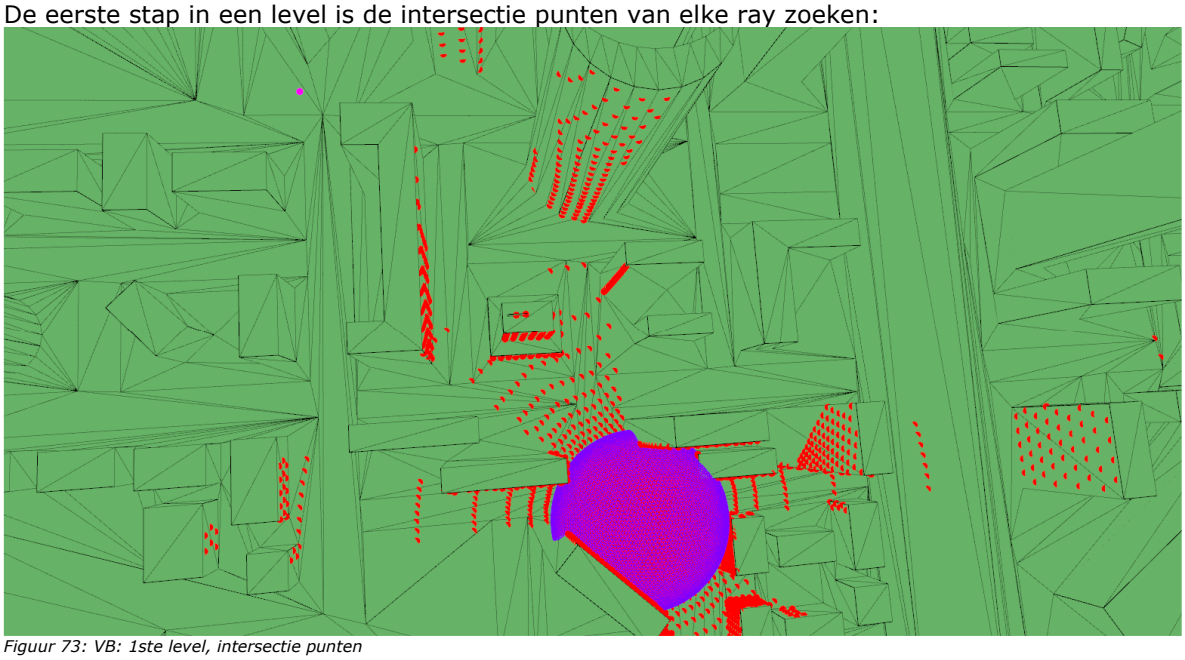
Verder zijn volgende eigenschappen ingesteld:

- de Keller Cones worden één keer opgesplitst, zowel de bronlijn als de reflectielijn.
- de beams worden niet opgesplitst als ze op een grenslijn botsen.



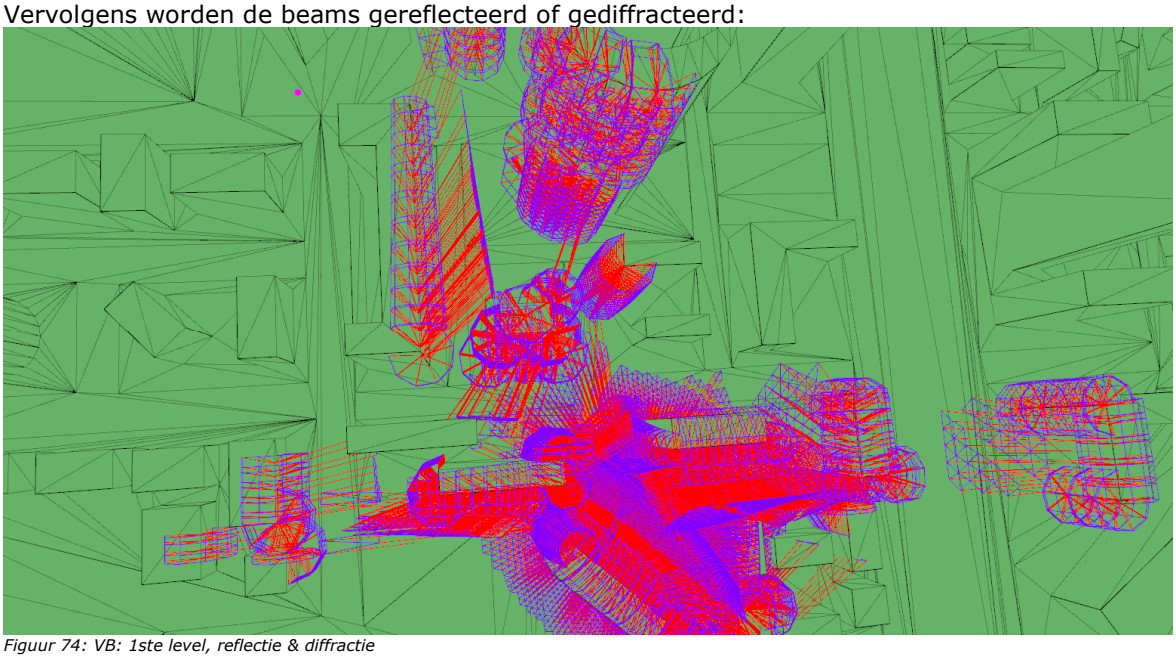
Figuur 72: VB: Virtueel model

***Eerste** level:*
De eerste stap in een level is de intersectie punten van elke ray zoeken:



Figuur 73: VB: 1ste level, intersectie punten

Vervolgens worden de beams gereflecteerd of gediffracteerd:

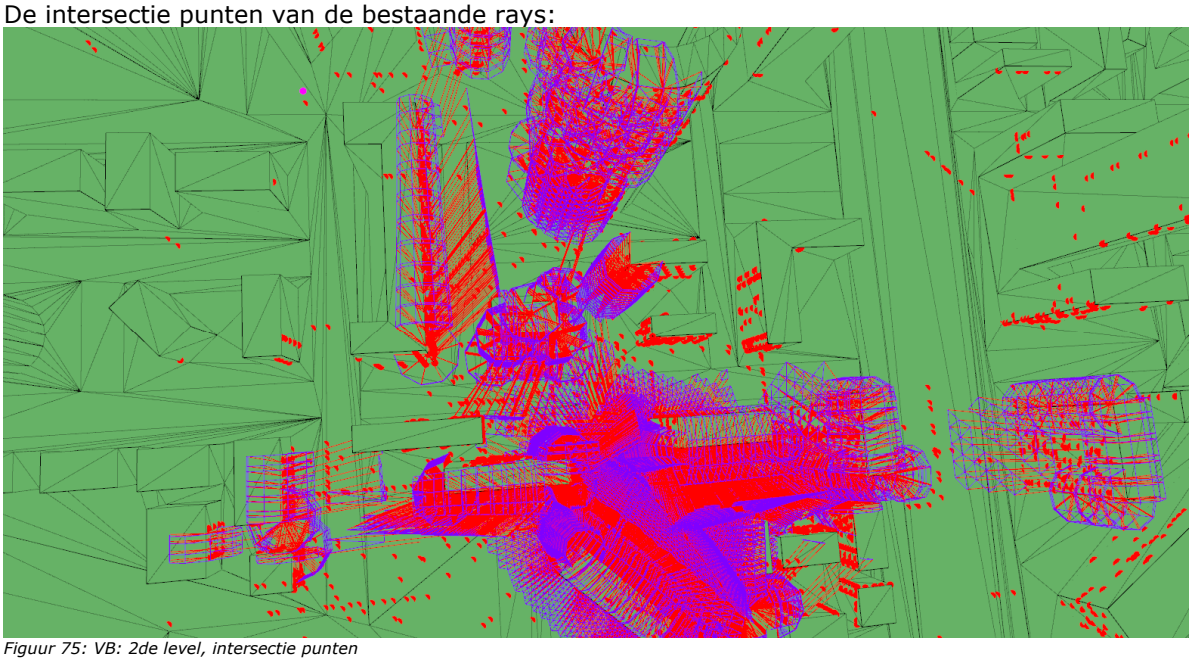


Figuur 74: VB: 1ste level, reflectie & diffractie

In Figuur 73 is te zien dat er veel rays botsen op het rechtse gebouw. Sommige beams botsen volledig, andere botsen op de grenslijn van het gebouw. In Figuur 74 is dan ook te zien dat er beams zijn die gereflecteerd worden en andere die een Keller Cone vormen.

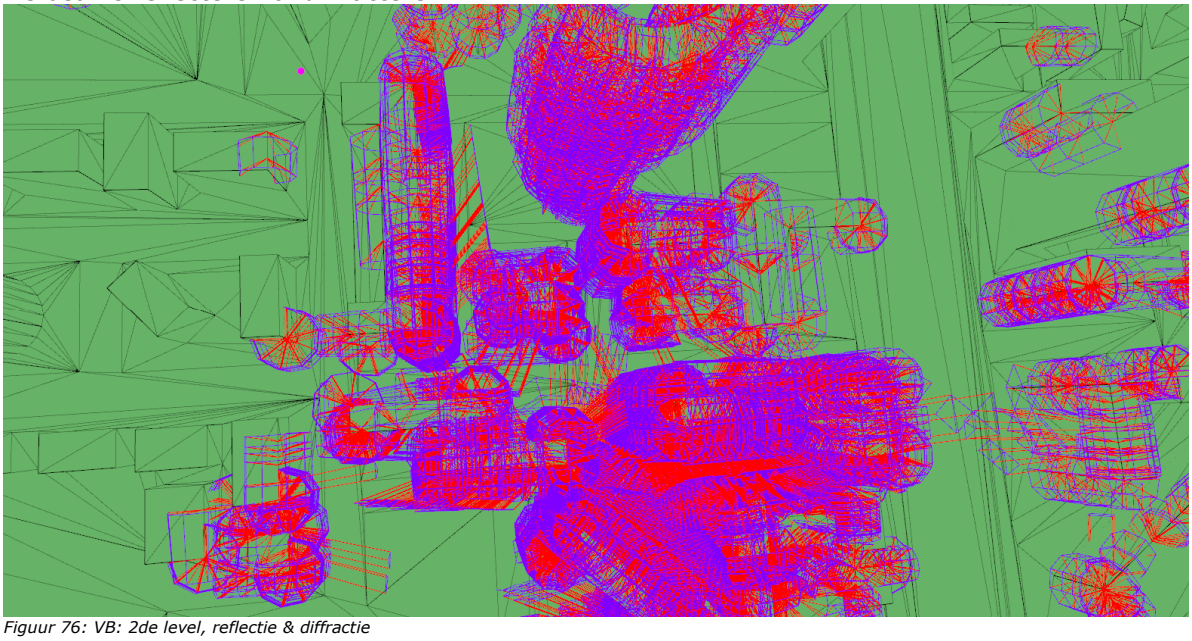
Verder is er in Figuur 73 ook te zien dat er bijna geen enkele beam volledig botst op het cilindervormig gebouw (bovenaan op de figuur). Het gebouw staat daarom ook vol met Keller Cones, waarvan sommige creeping beams genereren.

***Tweede** level:*
De intersectie punten van de bestaande rays:



Figuur 75: VB: 2de level, intersectie punten

De beams reflecteren of diffracteren:

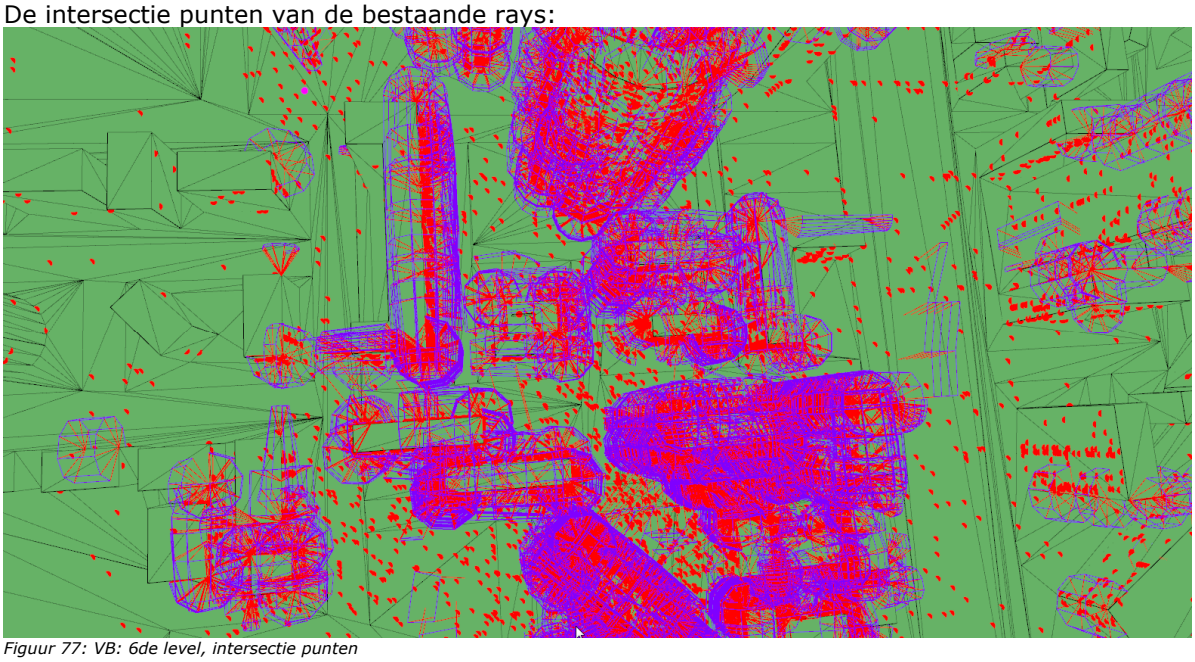


Figuur 76: VB: 2de level, reflectie & diffractie

In bovenstaande figuren is te zien dat de beams zich geleidelijk aan verder uitspreiden ten opzichte van het eerste level. Neem bijvoorbeeld de gebouwen rechtsboven op de figuur, deze zijn niet rechtstreeks bereikbaar vanuit de geluidsbron, na één level botsen enkele beams toch op het gebouw.

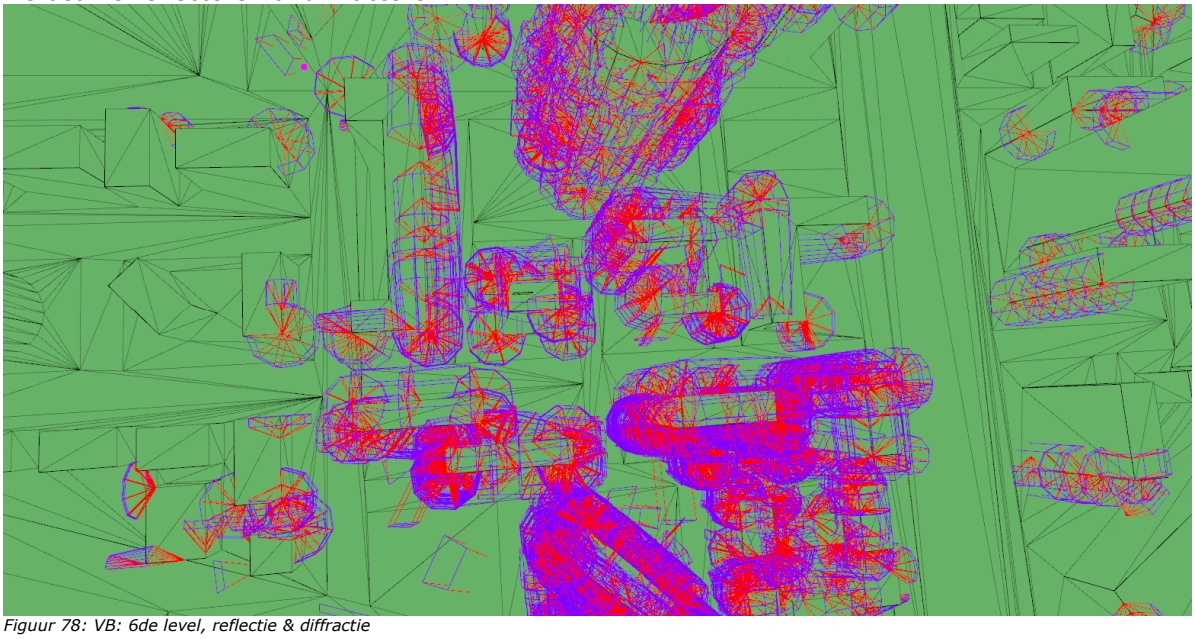
***Zesde** level:*

De intersectie punten van de bestaande rays:



Figuur 77: VB: 6de level, intersectie punten

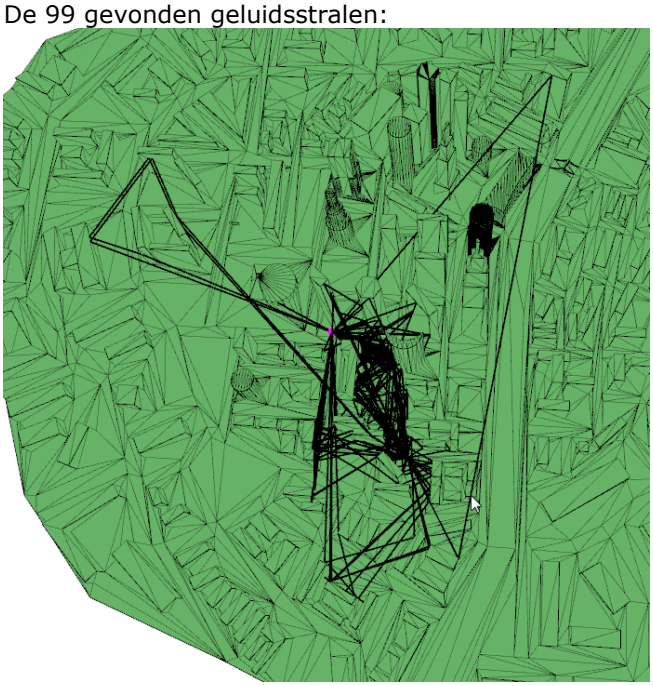
De beams reflecteren of diffracteren:



Figuur 78: VB: 6de level, reflectie & diffractie

Het zesde level bevat uiteindelijk 20680 beams en er werden 99 geluidsstralen gevonden van de geluidsbron naar het fieldpoint. Weet dat de bovenstaande figuren maar een klein gedeelte van het model tonen, na zes levels zijn er al meerdere beams verder geraakt dan de gebouwen die hierboven werden getoond.

De 99 gevonden geluidsstralen:

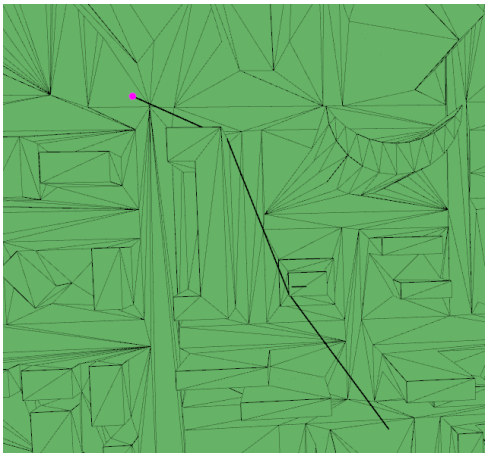


Figuur 79: VB: Verzameling geluidsstralen

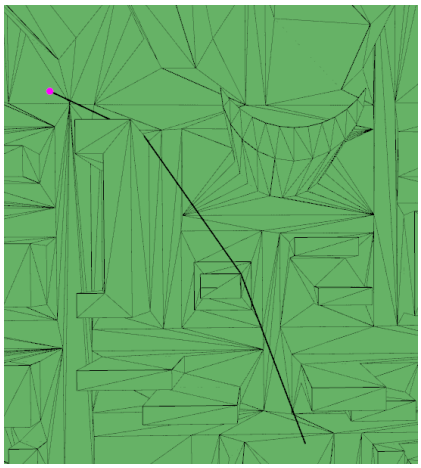
Houd er rekening mee dat **alle** geluidsstralen van geluidsbron naar fieldpoint worden berekend, er wordt dus geen rekening gehouden met de intensiteit van een geluidsstraal. Enkele getekende geluidsstralen zullen in realiteit geen intensiteit hebben.

Toch zou het kunnen dat de geluidsstraal die botst met het gebouw rechts boven een belangrijke meerwaarde heeft. Stel bijvoorbeeld dat alle reflectie van de geluidsstraal gebeuren op een materiaal dat een goede reflectie coëfficiënt heeft (bijvoorbeeld glas, wat veel voorkomt bij sky scrapers).

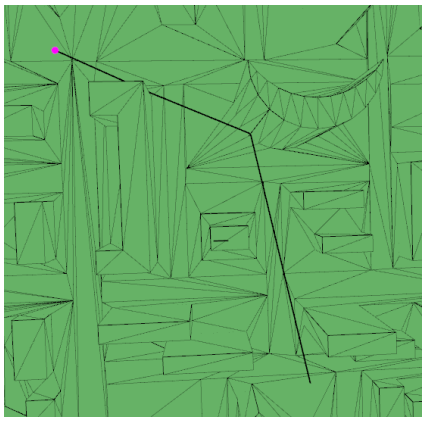
Enkele afzonderlijke geluidsstralen met de opsomming van effecten waaraan de geluidsstraal onderhevig is:



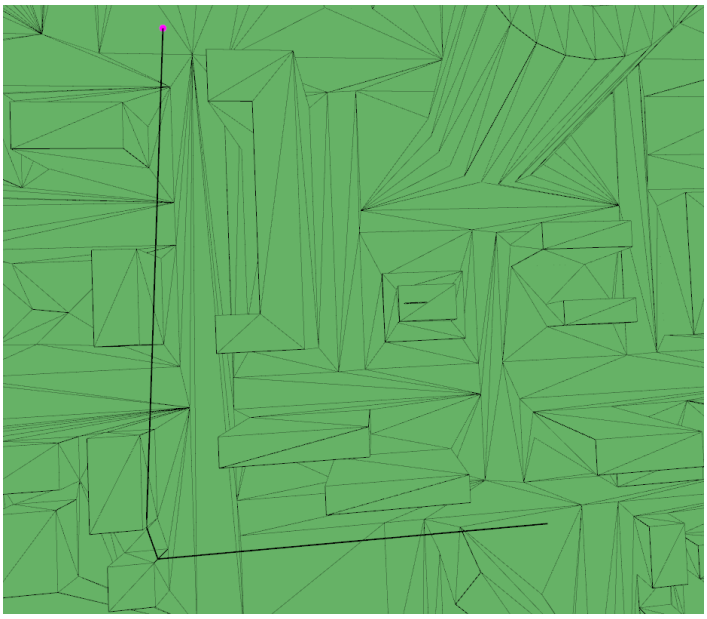
2 x forward diffractie



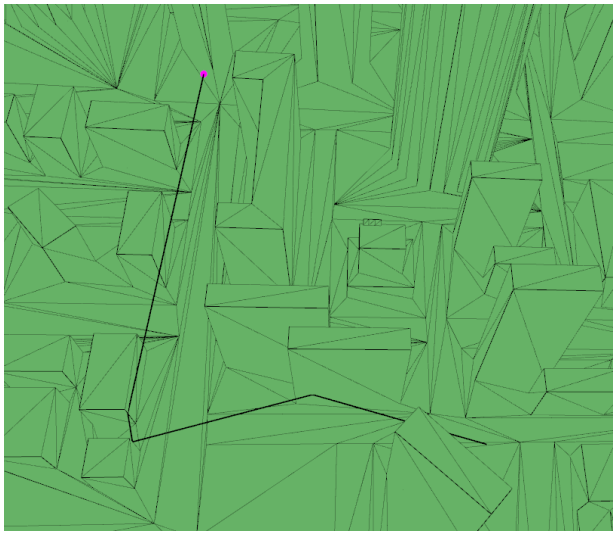
2 x forward diffractie



backward diffractie
& forward diffractie



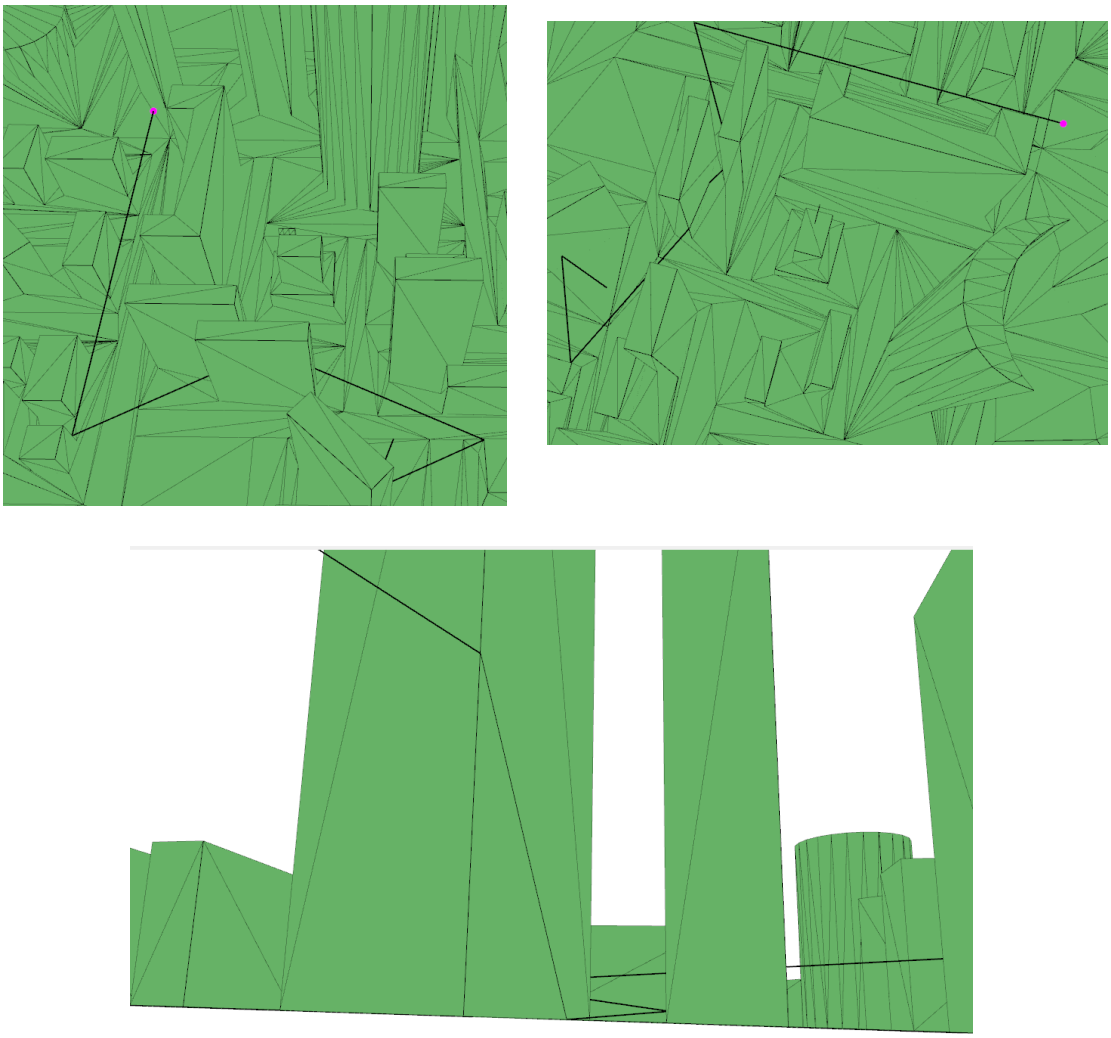
2 x forward diffractie



reflectie
& 2 x forward diffractie

Figuur 80: VB: Geluidsstralen

De geluidsstraal, geïllustreerd hieronder, bevat een creeping beam. De creeping beam kruipt op de voorgevel van een gebouw, zoals te zien is in de onderste afbeelding.



Figuur 81: VB: Geluidsstraal met creeping beam

De geluidsstraal is uiteindelijk onderhevig aan volgende akoestische effecten:

reflectie & backward diffractie & creeping beam & reflectie & backward diffractie

13. Bibliografie

[1] Foley, VanDam, Feiner & Hughes. *Computer Graphics: Principles and Practice.*

[2] Massal G.. *A raytracer in C++ - Introduction – What is ray tracing?* Opgevraagd op 16/11/2010 via http://www.codermind.com/articles/Raytracer-in-C++-Introduction-What-is-ray-tracing.html

[3] Lauterbach C., Chandak A. & Manocha D.. *Interactive sound propagation in dynamic scenes using frustum tracing.*

[4] *OpenGL Tutorials by NEHE*. Opgevraagd op 22/02/2011 via http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=08

[5] Waite group Press. *OpenGL Super Bible*. Opgevraagd op 22/02/2011 via http://opengl.czweb.org/ewtoc.html

[6] Chandak A., Lauterbachm C., Taylor M., Ren Z. & Manocha D. (2007). *AD-Frustum: Adaptive Frustum Tracing for Interactive Sound Propagation.*

[7] Noe N., Gaudaire F., Jean P. & Vermet M.. *A general ray-tracing solution to reflection on curved surfaces and diffraction by their bounding edges.*

[8] Drumm I.A. & Lam Y.W. (1999). *The adaptive beam-tracing algorithm.*

[9] *Dot Product*. Opgevraagd op 28/02/2011 via http://en.wikipedia.org/wiki/Dot_product

[10] *Cross Product*. Opgevraagd op 28/02/2011 via http://en.wikipedia.org/wiki/Cross_product

[11] *Icosahedron*. Opgevraagd op 01/03/2011 via http://en.wikipedia.org/wiki/Icosahedron

[12] Lewers T. (09/12/1991). *A Combined Beam Tracing and Radiant Exchange Computer Model of Room Acoustics.*

[13] Van Maercke D., Martin J.. *The prediction of echograms and impulse responses with the epidaure software.*

[14] Weisstein, Eric W. *Point-Plance Distance*. Opgevraagd op 09/03/2011 via http://mathworld.wolfram.com/Point-PlaneDistance.html

[15] Conil E. (10/11/2005). *Propagation électromagnétique en milieu complexe.*

[16] Vermet M., Noe N., Vauzelle R., Pousset Y. & Combeau P.. *Using asymptotic methods to compute diffracted pressure by curved surfaces.*

[17] Keller J.B. (13/09/1961). *Geometrical Theory of Diffraction.*

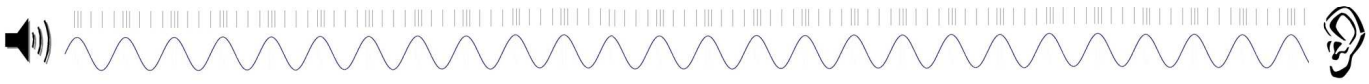
[18] Tsingos N., Funkhouser T., Ngan A., Carlbom I.. *Modeling Acoustics in Virtual Environments Using the Uniform Theory of Diffraction.*

[19] Sunday D.. *Distance between Lines and Segments with their Closest Point of Approach*. Opgevraagd op 18/04/2011 via http://softsurfer.com/Archive/algorithm_0106/algorithm_0106.htm

[20] Bikker J.. *Kd-Trees and More Speed.* Opgevraagd op 21/05/2011 via http://www.flipcode.com/archives/Raytracing_Topics_Techniques-Part_7_Kd-Trees_and_More_Speed.shtml

The quest for optimal & accurate sound virtualization:

Ray-acoustic modeling based on Adaptive Beam Tracing



Geometrical Acoustics – the evolution

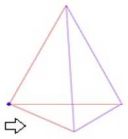
Image Source Method (ISM) → Ray Tracing Method (RTM) → **Adaptive Beam Tracing Method (ABTM)**

Geometrical Acoustics approaches are derived using asymptotic **approximations** of the wave equation in the infinite frequency limit. Their basic assumption is that sound propagates in linear paths, like light. The first GA approaches being investigated were the Image Source Method (MISM) and the Ray Tracing Method (RTM).

A more recent GA approach is the **Adaptive Beam Tracing Method** (ABTM). The basic principle of the ABTM is to launch beams as bundles of rays that have as purpose to find all visible sound receivers. These sound receivers and their associated ray paths represent the physical events that an acoustic wave can undergo, such as specular reflections against physical boundaries, but also diffraction events around edges. In the end, the pressure at a given receiver location is simply the sum of all ray path contributions. Calculation equations that describe the energy contributions in function of frequency exist for local physical events like specular reflections and edge diffractions . Appealing in this ray approach is that a ray path is independent of any wavelength compliant with the asymptotic high frequency condition. The challenge however lies in **finding all possible combinations** of these events, e.g. a wave can first reflect against a surface, then diffract around a sharp edge, then diffract around a curved surface, then reflect against another surface, etc.

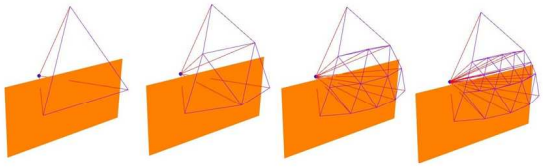
1 Emission of sound

Beam



A beam is defined by its rays. In this picture the beam starts from a point, the rays are colored red and the beam is colored blue.

Adaptive Beam

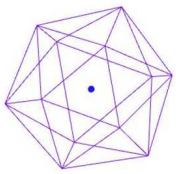


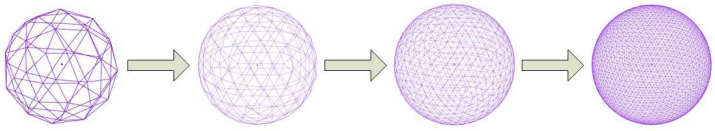
The purpose of the adaptive beam tracing method is to show how a beam adapts to an intersecting wall. As shown in the first picture, the beam has 2 rays intersecting a wall. The other ray is emitted into the infinity. What will happen with this beam? Will it reflect on the wall, be emitted into the infinity or will it diffract? The answer is all 3, but executed on all different and smaller beams. The different beams are made by subdividing the original beam into 4 smaller beams. The subdivision continues until a maximum level is reached.

Sound source: ICOSAHEDRON

A sound source is virtualized by a regular polytope, which is a spatial figure composed of regular facets and regular vertex figures.

Since the beams are triangles, a **icosahedron** is used as regular polytope. A standard icosahedron has 20 identical equilateral triangular faces, 30 edges and 12 vertices.



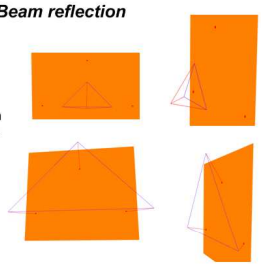


For extra accuracy the standard icosahedron is subdivided. Again every beam is subdivided into 4 new and smaller beams.

2 Physical events

Beam reflection

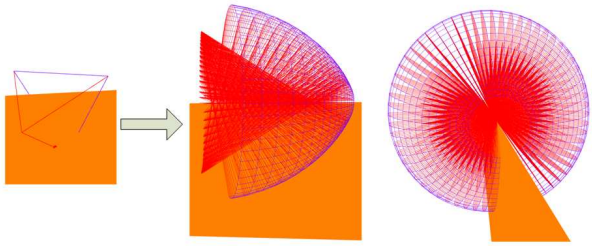
As mentioned above, a beam is defined by its rays. Therefore, a beam will only reflect when all his rays intersect with a wall.



Diffraction: KELLER CONE

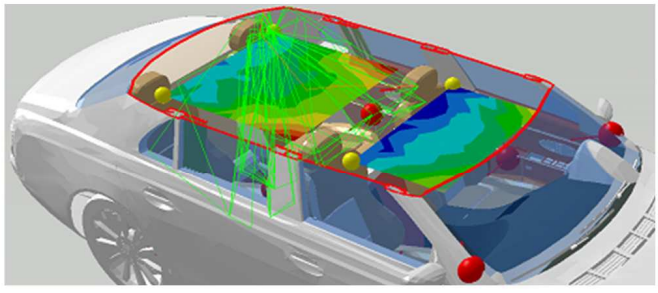
Diffraction is a physical event occuring when a sound wave intersects with the side of a wall. New sound waves will then bend around the intersecting wall, making a distinct pattern for new sound waves. This pattern is called "**the keller cone**".

The first picture at the right, shows a beam, assuming the maximum level of subdivision is reached, with 1 intersecting ray on a wall and the other rays are emitted into infinity. This beam is a typical case of the physical diffraction event. The keller cone defines new beams that **bend around the wall**.



The keller cone defines **astigmatic beams**, defined by its source – and reflection line. Astigmatic beams have a quadrilateral face, unlike the standard beam that has a triangular face with a source point.

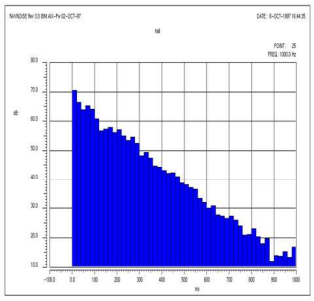
3 The result!!



The picture on the left shows the interior of an existing Mercedes Benz model. Sound waves are emitted from the center speakers and sound paths are shown to the left passenger on the backseat.

Most sound paths are created by **reflected sound waves**. The intensity of a reflected sound wave is depending on the material the sound wave gets reflected on. In this example the car constructor can easily experiment with different materials to obtain the most accurate sound quality. The echograms show the intensity of the combined sound emissions from the speakers.

Another requirement to get optimal sound is minimizing the sound echoes. A big construction with good reflecting materials (for example a church) has many echoes which will give a distorted sound result. The histogram on the right shows an example of some echoes. Sound with an intensity of +/- 70dB is received at the start of the measurement. After 1 second there is still some sound received with an intensity of +/- 17dB.



LMS ENGINEERING INNOVATION

External promotors: Massa Gert, Manu De Geest

Van Auseloos Christophe

http://vanauseloos.be/masterproef

HOGESCHOOL GENT
LID VAN DE ASSOCIATIE UNIVERSITEIT GENT

Internal promotor: Van De Wiele Koen