# Visualisering av elektronstrukturer

# Visualisering av volymsdata

**Andreas Kempe och Viktor Bernholtz**

**Version 0.4**

Status

| | | |
|---|---|---|
| Granskad | VB | 2018-05-25 |
| Godkänd | | |

# PROJEKTIDENTITET

2018/VT, Grupp 2
Linköpings Tekniska Högskola, IFM

Gruppdeltagare


| **Namn** | **Ansvar** | **Telefon** | **E-post** |
|---|---|---|---|
| Andreas Kempe | Sekreterare (SE) | 073-9796689 | andke133@student.liu.se |
| Viktor Bernholtz | Viktor Bernholtz (VB) | 073-0386030 | vikbe253@student.liu.se |

**Kund**: IFM, Linköpings universitet, 581 83 Linköping
**Kontaktperson hos kund**: Rickard Armiento, 013-281249, rickard.armiento@liu.se
**Kursansvarig**: Per Sandström, 013-282902, persa@ifm.liu.se
**Handledare**: Johan Jönsson, 013-281176, johan.jonsson@liu.se

# Innehåll

# Figurer

# Tabeller

Dokumenthistorik

| **Version** | **Datum** | **Utförda förändringar** | **Utförda av** | **Granskad** |
|---|---|---|---|---|
| 0.4 | 2018-05-25 | En del korrigeringar efter återkoppling från beställare. | AK, VB | VB |
| 0.3 | 2018-05-14 | En del korrigeringar efter återkoppling från handledare. | AK, VB | VB |
| 0.2 | 2018-05-03 | Korrigering av språk efter återkoppling från Tema. | AK, VB | AK, VB |
| 0.1 | 2018-04-27 | | AK, VB | AK, VB |

# 1 Inledning

## 1.1 Bakgrund

Inom många olika fält undersöks volymer och deras innehåll. Ett välkänt område är MR, magnetisk resonanstomografi, där kroppen utsätts för starka elektromagnetiska fält som gör att olika vävnader i kroppen kan läsas av [14]. Ett annat område är beräkningar gjorda med hjälp av kvantmekanik på olika kristallstrukturer. Mjukvaran VASP kan bland annat användas till att beräkna laddningstätheten i en kristall och resultatet fås som laddningstätheten för olika punkter i kristallens volym [1].

Det är svårt att bilda sig en uppfattning om vad en stor mängd siffror kopplade till koordinater betyder utan att åskådliggöra resultatet på ett lättfattligt sätt. Vid Linköpings universitet används en mjukvara vid namn Inviwo som utvecklas av forskargruppen *Scientific Visualization Group* vid Linköpings universitet i Norrköping [23]. Den klarar av att visualisera data på en mängd olika sätt, bland annat genom volymsrendering [12].

## 1.2 Syfte

Arbetet ämnar göra en undersökning av hur ett par metoder som används i mjukvaran Inviwo fungerar för att sedan göra en jämförelse av deras resultat och prestanda.

## 1.3 Frågeställning

Det här arbetet ämnar besvara förljande frågor: Hur fungerar volymsrendering? Hur fungerar ytextraktion med marching cubes? Hur skiljer sig prestandan och resultat mellan de två metoderna?

## 1.4 Avgränsningar

Arbetet avgränsas till att undersöka volymsrendering genom strålföljning och extrahering av isoytor med hjälp av algoritmen *Marching Cubes*. Detta då dessa två metoder är aktuella för kandidatarbetet *Visualisering av elektronstrukturer* som rapporten skrivs för.

## 1.5 Definitioner

**Gradient** betecknas $\nabla f$ och är en vektor som har de partiella derivatorna till en funktion $f$ som sina komponenter. Detta leder till att gradienten pekar i den riktning för vilken funktionen i fråga har störst ökning [16].

**Implicit funktion** definieras i nationalencyklopedin enligt följande: "implicit funktion, i matematiken en funktion som defineras indirekt genom en eller fler ekvationer som funktionsvärdena skall uppfylla."[17].

**Interpolering** är inom matematiken en metod att approximativt beräkna mellanliggande variabelvärden utgående från en funktion där bara vissa värden är kända [18].

**Inviwo** är en mjukvara för visualisering av data som utvecklas vid Linköpings universitet [23].

**Isoyta** är den yta som defineras av den implicita funktionen.

**Isovärde** är ett värde på isoytan.

**Sampel** se *Sampling*.

**Sampling** är en process där diskreta mätningar görs på en storhet och ett mätvärde, s.k. sampel, erhålles [19].

**Skalär** är en storhet med ett värde som saknar riktning [24].

**Skalärfält** är det fält som knyter samman skalärer med punkter i ett rum [22, s. 6].

**Strålgång** är den väg ljuset tar genom ett objekt.

**Tetraeder** är en geometrisk kropp som begränsas av fyra triangelformade plana ytor [20].

**Uppslagningstabeller** används för data som programmet ofta efterfrågar, i vårt fall matematiska tabeller. Dessa används för att det går fortare att slå upp ett värde än att räkna ut det [4].

**Volumetrisk** betyder att någonting kan betraktas i tre faktiska dimensioner. Det betyder att vi kan titta på någonting från alla håll eller rotera på det [5].

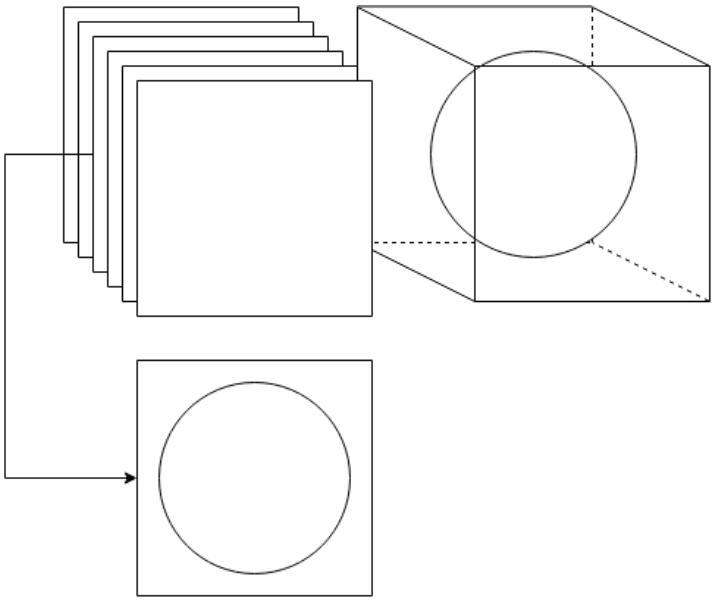
**Voxel** är ett indivuduellt volymelement, precis som pixel är ett indivuduellt bildelement [10].

**Överföringsfunktion** är ett sätt att representera ett linjärt tidsinvariant system med avseende på in-utförhållande [15].

# 2 Volymsrendering

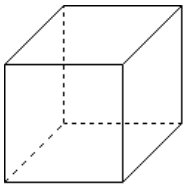
Volymsrendering är en teknik som tillåter användaren att se in i 3D-objekt. Volymsrendering kan göra delar av objektet transparenta och andra icke transparenta, det går ofta att bestämma nivå av transparens och färg [11].

Metoden genererar bilder av 3D-volymdata utan att explicit extrahera geometriska ytor från datan. Den här tekniken använder en optisk modell för att koppla data till optiska egenskaper så som färg och transparens. Under renderingen skapas de optiska egenskaperna längs varje strålgång för att sedan skapa en bild av datan. Även om datan här tolkas som en kontinuerlig funktion i rymden så representeras den, av praktiska skäl, som en uppsättning likformiga 3D-matriser. I grafikminnet sparas sedan volymdata i antingen en stack av 2D-bilder eller som ett enda 3D-objekt. I figur 1 ses en stack av 2D-bilder till vänster som är tvärsnitt av 3D-bilden till höger.

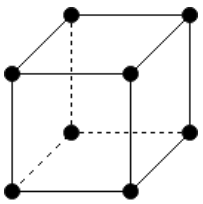


Figur 1: Grov skiss över 2D-stack och 3D-objekt

En 3D-bild byggs sedan upp av voxlar, där varje voxel har en specifik plats i rymden och har ett eller flera datavärden knutna till sig, se figur 2 respektive figur 3. Datavärden som ligger mellan två specifika platser i rymden erhålls med hjälp av interpolering med närmsta granne. Denna process kallas rekonstruktion och är en viktig del i volymsrendering.



Figur 2: Lådan är en voxel med ett enda värde



Figur 3: Lådan är en voxel med fler värden knutna till sig

Det huvudsakliga syftet med den optiska modellen är att beskriva hur partiklarna i en volym interagerar med ljus. I den vanligaste, lite enklare modellen antas volymen innehålla partiklar som samtidigt emitterar och absorberar ljus. De lite mer komplexa modellerna har lokala ljuskällor, volumetriska skuggor och tar hänsyn till hur ljuset sprids. De optiska egenskaperna anges antingen direkt av datavärden eller så beräknas de med en eller fler överföringsfunktioner. Överföringsfunktionerna har som mål att tydliggöra eller klassificera data som är av intresse för modellen. Då komplexa överföringsfunktioner tar tid att beräkna slås de ofta upp i uppslagningstabeller, medan de enklare funktionerna kan beräknas direkt.

Bilden skapas sedan genom att sampla volymen längs alla strålgångar och tillsätta de optiska egenskaperna. För den vanliga emissions-absorptionsmodellen som nämnts ovan används ekvation (1) för att beräkna färg och ekvation (2) för att beräkna transparens.

$$C = \sum_{i=1}^{n} C_i \prod_{j=1}^{i-1} (1 - A_j) \tag{1}$$

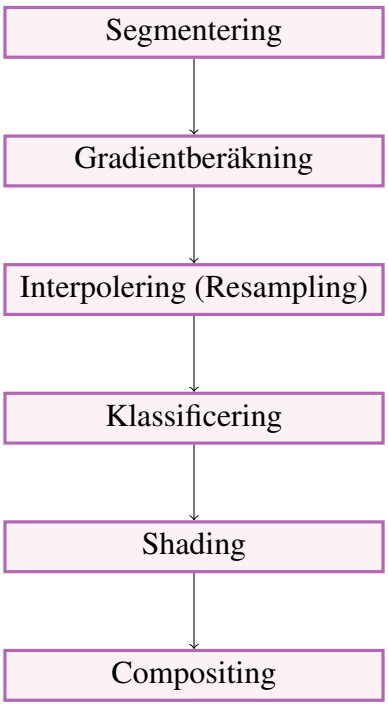
$$A = 1 - \prod_{j=1}^{n} (1 - A_j) \tag{2}$$

$C_i$ och $A_j$ är färgen respektive transparensen som överföringsfunktionen tilldelar datavärdet för sampel i och j. Transparensen $A_j$ approximerar absorptionen och den transparensviktade färgen $C_i$ approximerar emissionen och absorptionen längs strålgången mellan sampel $i$ och $i + 1$. $C_i$ representerar alltså den mängd ljus som emitteras vid ett sampel $i$ och hur mycket ljuset dämpas innan det når betraktaren. Både ekvation (1) och (2) löses effektivt genom att sortera alla sampel längs strålgången och iterativt beräkna både färgen och transparensen [10].

I avsnitt 2.1 nedan beskrivs de olika stegen i denna process.

## 2.1 Renderingspipeline

En volymsrendering har, som tidigare nämnts, som mål att ta en mängd data bestående av voxlar i 3D-rymden för att skapa en projektion av dessa till en 2D-bild. För att åstadkomma detta behöver ett antal problem lösas. I boken *Introduction To Volume rendering* beskrivs en renderingspipeline i sex steg, illustrerat av figur 4 [13, s. 29].




Figur 4: Renderingspipeline för volymsrendering. [13, s. 29]

I det första steget, segmentering, märks datapunkterna för att avgöra vad olika voxlar representerar i materialet. Detta då voxlarna i sig bara utgör delar av rymden. De kan till exempel representera en viss densitet, ett visst material eller vilken annan storhet eller egenskap som helst som går att associera till en punkt i rummet [13, s. 29-30].

Gradientberäkningssteget går ut på att hitta gradienter mellan de olika materialen. En gradient är en vektor som pekar i den riktning som har störst förändring och på så vis kan övergångar mellan olika material hittas. Denna information används senare i klassificerings- och shadingsteget [13, s. 30].

När strålar skickas genom volymen för att sampla dess data leder det vanligtvis till att strålen missar voxlar och passerar mellan dem. I interpoleringssteget löses problemet genom att, för varje punkt, interpolera dess värde från de kringliggande voxlarnas värden [13, s. 30].

Stegen klassificering och shading färglägger respektive ljusätter punkterna i volymen. Klassificeringen utgår från de etiketter som tilldelats punkterna i segmenteringssteget och färglägger dem baserat på vilken typ de tilldelats, varefter shadingsteget gör beräkningar på ljussättning av desamma [13, s. 30]. Den matematiska beskrivningen hittas tidigare i avsnitt 2, ekvation 1 och ekvation 2.

Slutligen nås compositingsteget under vilket datan reduceras. Efter att ha interpolerat datavärden längs en stråle finns det i regel en väldigt stor mängd datapunkter som måste reduceras till en enskild punkt på 2D-ytan som volymen projiceras på. I detta steg gås punkterna igenom och ett enskilt värde för varje pixel i bilden beräknas [13, s. 30]. En enkel metod för att bestämma färgen på pixeln är att översätta värdet för den voxel längs strålen som har störst värde till en färg, vilket kan ge en tydlig bild av isolerade strukturer [13, s. 49-50].

## 2.2 Pathtracing

I volymsrenderingssammanhang finns det en metod som heter pathtracing. Det är en strålföljningsmetod och den går ut på att strålar skickas ut från pixlarna, vilka bygger upp 2D-bilden, genom den volym som skall renderas. Längs med dessa strålar samplas volymens voxlar och deras värden interpoleras för att sedan kopplas till färger i slutrenderingen. Denna process sker i stegen interpolering, klassificering, shading och compositing i renderingspipelinen som kan ses i figur 4 och beskrivs övergripande i avsnitt 2.1 [13, s. 58-59].

### 2.2.1 Matematisk beskrivning

En transformation är en process som tillåter att en uppsättning värden tas från ett rum till ett annat och kan då definieras matematiskt som $T : U \rightarrow V$. Där funktionen $T$ transformerar punkter från rummet $U$ till rummet $V$ [21, s. 150-151].

Inom linjär algebra är det vanligt att uttrycka transformationer medelst matriser, vilket görs enligt uttrycket i (3) som är ett exempel på en transformation i tre dimensioner. Transformationer i högre antal dimensioner uttrycks lätt på samma sätt genom att lägga till fler rader och kolumner i matriserna [21, s. 151].

$$T_A(X) = AX = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} ax + by + cz \\ dx + ye + fy \\ gx + hy + iz \end{bmatrix} \tag{3}$$

I fallet med volymsrendering skall voxlarna, som tidigare nämnts, tas från volymsrummet i tre dimensioner till det tvådiminsionella rum som den önskade bilden utgör. För att göra detta används en transformationsmatris som utför den önskade projektionen [13, s. 52].

### 2.2.2 Homogena koordinater

En rotation, skalning och translation kan i tre dimensioner representeras av en 4x4-matris. För att åstadkomma detta läggs en fjärde koordinat till de tre existerande rumskoordinaterna och en punkt representeras således av koordinaterna $(x, y, z, w)$, vilket då kallas homogena koordinater. Punkten $(0, 0, 0, 0)$ är inte tillåten, utan åtminstone en homogen koordinat måste vara nollskild. Normalt sätts $w \neq 0$ och punkten $(x, y, z, w)$ kan då uttryckas som $(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}, 1)$, vilket kallas de kartesiska koordinaterna för den homogena punkten. Vanligtvis utförs divisionen med $w$ om $w \neq 1$ [8, s. 204].

När en punkt nu representeras av de fyra homogena koordinaterna kan därför en translation av punkten $(x, y, z, 1)$ till $(x + d_1, y + d_2, z + d_3, 1)$ beskrivas med följande matrisekvation:

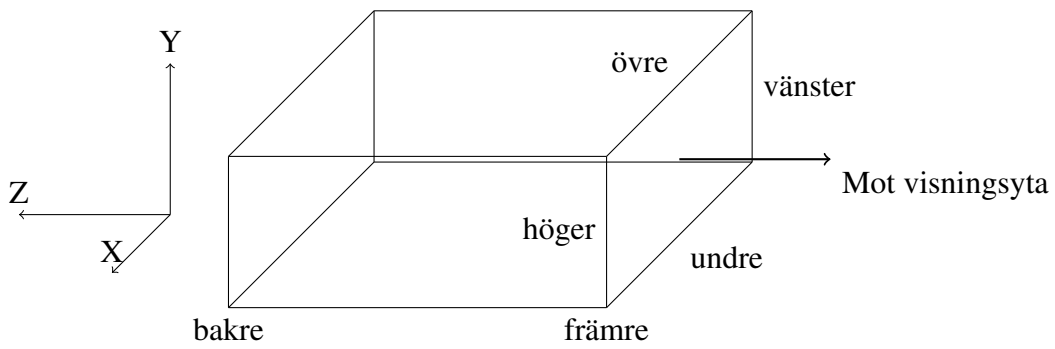
$$\begin{bmatrix} 1 & 0 & 0 & d_1 \\ 0 & 1 & 0 & d_2 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + d_1 \\ y + d_2 \\ z + d_3 \\ 1 \end{bmatrix}$$

Värt att notera är att $w = 1$ även efter multiplikationen och den nya punkten således har rört sig inom samma tredimensionella volym i det fyrdimensionella rummet. Detta då varje $w$-koordinat ger ett nytt tredimensionnelt rum. Trippeln $(x + d_1, y + d_2, z + d_3)$ har alltså utfört en enkel translation jämfört med $(x, y, z)$.

### 2.2.3 Ortografisk projektion

En ortografisk projektion är en projektion där strålarna sänds in parallellt över hela bildytan. Därmed formar volymen som strålarna passerar ett rätblock, se figur 5, och är en enklare projektion än perspektivistiska projektionen, se avsnitt 2.2.4, som tar hänsyn till betraktarens perspektiv när bilden projiceras [13, s. 54-55].

Som en följd av att strålarna skickas parallellt genom volymen, i form av ett rätblock, bevaras storleken för alla objekt i volymen för en ortografisk projektion [13, s. 54].




Figur 5: Illustration av en ortografisk projektionskub. [13, s. 55]

Den generella transformationsmatrisen för en ortografisk projektion ges av matrisen i ekvation (4), där beteckningarna hittas i figur 5, hämtad ur *Introduction To Volume rendering* [13, s. 54-55].

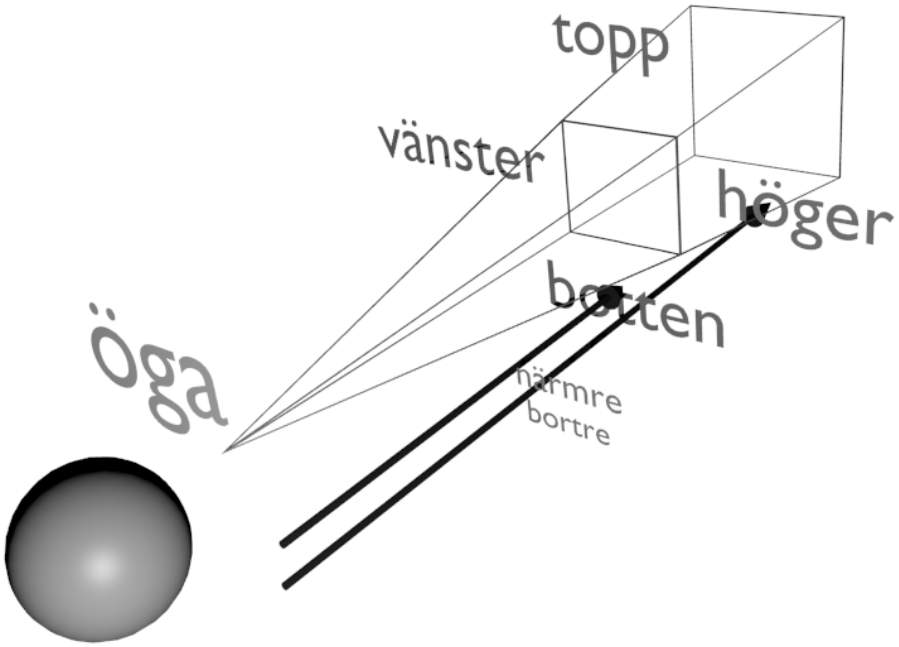
$$\begin{bmatrix} \frac{2}{\text{höger}-\text{vänster}} & 0 & 0 & t_x \\ 0 & \frac{2}{\text{övre}-\text{undre}} & 0 & t_y \\ 0 & 0 & \frac{-2}{\text{bakre}-\text{främre}} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

$$\begin{aligned} t_x &= \frac{\text{höger} + \text{vänster}}{\text{höger} - \text{vänster}} \\ t_y &= \frac{\text{övre} + \text{undre}}{\text{övre} - \text{undre}} \\ t_z &= -\frac{\text{bakre} + \text{främre}}{\text{bakre} - \text{främre}} \end{aligned}$$

### 2.2.4 Perspektivisk projektion

Den perspektiviska projektionen är en projektion som tar hänsyn till betraktarens perspektiv och leder till att objekt i förgrunden ser större ut än objekt som befinner sig längre bort [13, s. 52].

För att åstadkomma en perspektivisk projektion skickas strålarna genom volymen i form av en pyramid med avhuggen topp enligt figur 6.




Figur 6: Illustration av en perspektivisk projektionspyramid. [13, s. 53]

Även den perspektiviska projektionen uttrycks som en matris och har formen som hittas i ekvation (5).

$$\begin{bmatrix} \frac{2\times\text{närmre}}{\text{höger}-\text{vänster}} & 0 & A & 0 \\ 0 & \frac{2\times\text{närmre}}{\text{topp}-\text{botten}} & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{5}$$

$$A = \frac{\text{höger} + \text{vänster}}{\text{höger} - \text{vänster}}$$
$$B = \frac{\text{topp} + \text{botten}}{\text{topp} - \text{botten}}$$
$$C = -\frac{\text{bortre} + \text{närmre}}{\text{bortre} - \text{närmre}}$$
$$D = -\frac{2 \times \text{bortre} \times \text{närmre}}{\text{bortre} - \text{närmre}}$$

### 2.2.5 Model-view-matris

För att kunna utföra en projektion räcker det inte med en projektionsmatris, utan det krävs även en beskrivning av från vilket håll volymen betraktas. Denna beskrivning görs med en så kallad model-view-matris, vilken multipliceras med projektionsmatrisen för att få den fullständiga transformationen. Model-view-matrisen är en 4x4-matris som innehåller ett antal konstanter [13, s. 55-56]. Då model-view-matrisen enbart flyttar betraktningsvinkeln kommer den inte beskrivas närmare här.

### 2.2.6 Invers projektion

När transformationsmatriser som beskriver den önskade renderingen av bilden erhållits återstår problemet att beräkna transformen av den faktiska datan.

Pathtracing drar nytta av att transformationsmatriserna är inverterbara. Genom att invertera matrisen fås en transformation som översätter varje punkt på bildens 2D-yta till en punkt i 3D-rummet. På detta sätt behöver inte varje punkt i rummet gås igenom, utan det är möjligt att utgå från varje pixel i bilden och endast variera djupet, vilket motsvarar att skicka strålar in genom volymen [13, s. 57-59].

Inversen av model-view-matrisen kombinerat med projektionen beskrivs enligt ekvation (6), där varje element är namngivet med *I* på slutet för att indikera att det är den inversa matrisen. Notera således att t.ex. både $a = aI$ och $a \neq aI$ kan vara sant för olika inverser [13, s. 60].

$$\text{invers}\left(\begin{bmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & o \\ 0 & 0 & 0 & 1 \end{bmatrix}\right) = \begin{bmatrix} aI & eI & iI & mI \\ bI & fI & jI & nI \\ cI & gI & kI & oI \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

Genom att ta två punkter på bildens 2D-yta, $(x'', y'')$, och välja något $z''$ med $w = 1$, fås, genom att multiplicera med inversa transformen, homogena koordinater $(x', y', z', 1)$. Där $(x, y, z) = (x', y', z')$ är en punkt i volymen som önskas projiceras, se ekvation (7) [13, s. 60].

$$\begin{bmatrix} aI & eI & iI & mI \\ bI & fI & jI & nI \\ cI & gI & kI & oI \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x'' \\ y'' \\ z'' \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{7}$$

Genom att att variera $z''$ kan nu olika punkter i volymen samplas. Bilden byggs upp genom att variera värdet på $z''$ för alla $(x'', y'')$ på bildens 2D-yta [13, s. 60].

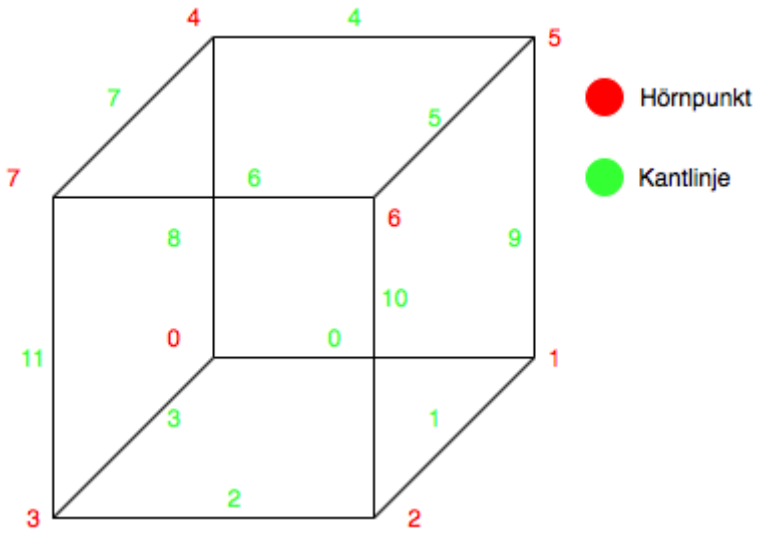
# 3 Marching cubes

Marching cubes är en algoritm för att skapa en isoyta utifrån volymetrisk data. Marching cubes kan appliceras inom ett flertal områden men de vanligaste är medicin och matematik. Inom medicin används det till exempel för att återskapa 3D-ytor av organ eller liknande och inom matematik för att skapa 3D-konturer av skalärfält [3].

Marching cubes kan defineras enligt följande: Givet ett objekt är det ett test för att avgöra om en godtycklig punkt ligger inom objektet samt inom vilka gränser objektet existerar. Dela in objektet i ett godtyckligt antal kuber. Testa om hörnen på varje kub är inuti objektet. För varje kub, där några hörn är inuti och några är utanför objektet, måste ytan passera genom kuben. Objektet skär alltså kubens kanter mellan hörnen av motsatt klassificering. Rita en yta i varje kub som förbinder dessa skärningar. Objektet är färdigt [2].

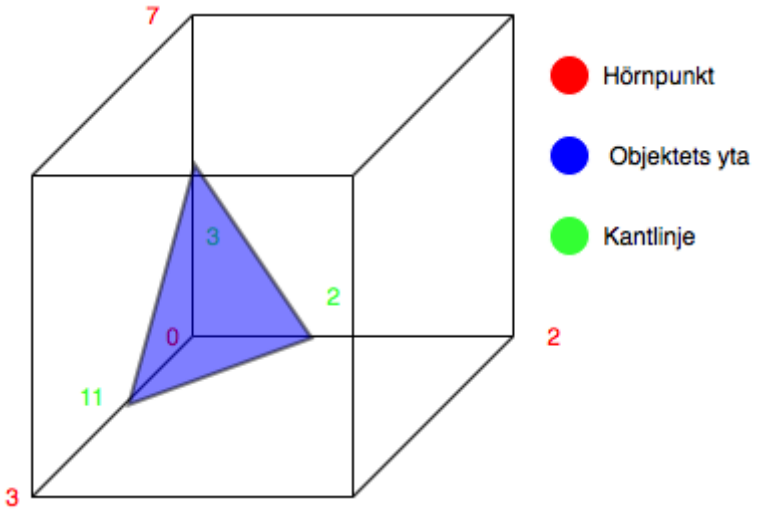
Marching cubes är alltså en algoritm för att skapa ett triangelnät från en implicit funktion. Det fungerar genom att iterera (engelska marching) över ett enhetligt kubiskt rutnät. Alla åtta hörnpunkter i kuben har ett binärt värde. Är alla hörnpunkter antingen ett eller noll kommer således ingen triangel att skapas, då hela kuben är under respektive över ytan. Är detta inte fallet kommer en eller flera trianglar blidas i kuben. Dessa trianglar kopplas sedan samman och bildar ytobjektet. I figur 7 kan vi se index för alla de åtta hörn algoritmen läser av värden på [7].

Figur 7 nedan, visar även kantlinjerna för en kub, det är alltså ett antal av dessa som skärs när en triangel skapas [3].



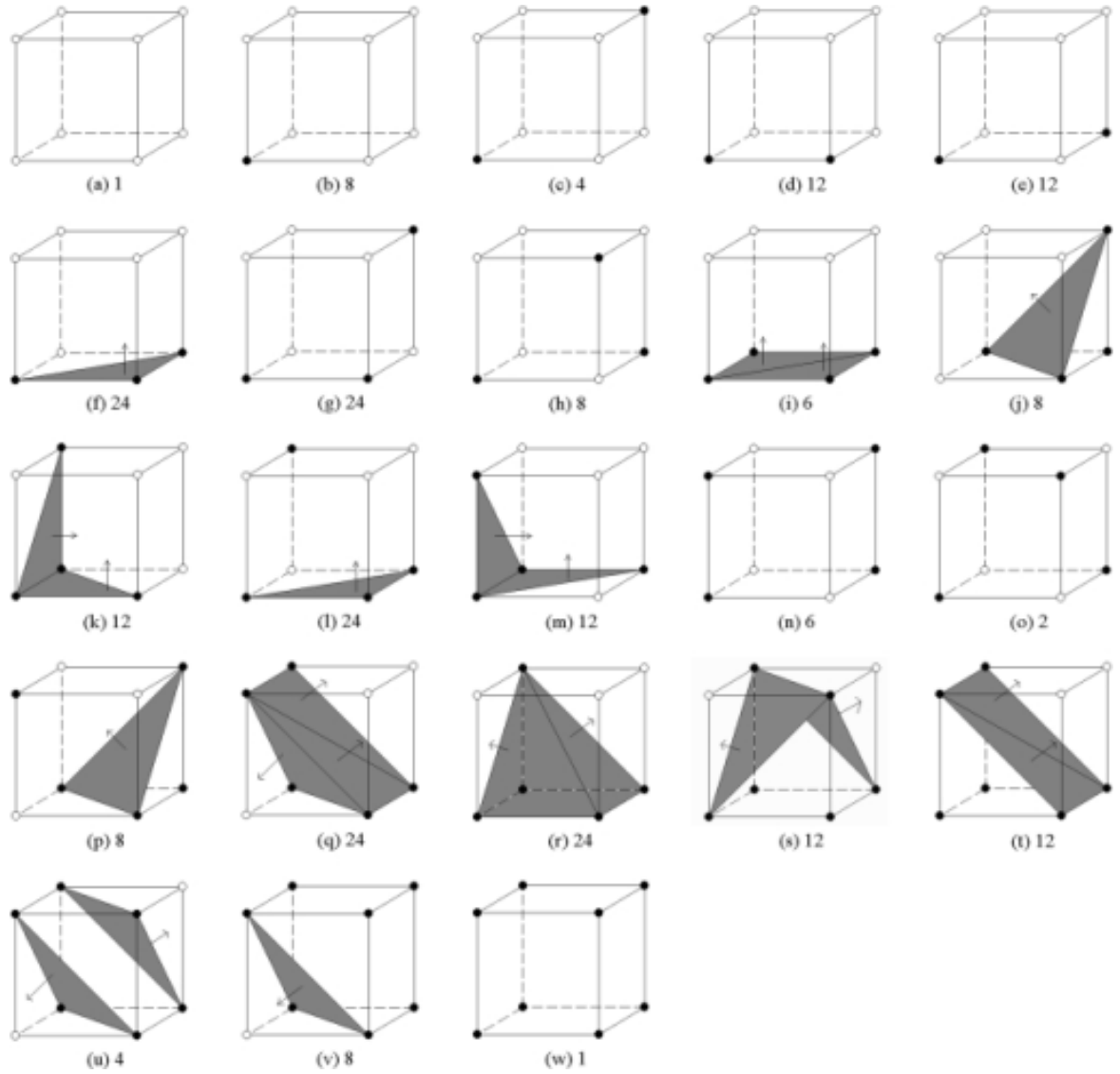

Figur 7: Indexering av hörn och kantlinjer i en kub. Inspirerad av Paul Brouke [3].

Ett exempel då en triangel skapas. Ett hörn är inuti objektet har värdet ett och om hörnet är utanför objektet har det värdet noll. Om alla hörn förutom hörn 0, har värdet noll kommer en triangel bildas enligt figur 8. I ett så här enkelt fall med har binära hörnpunkter kommer kantlinjerna, i detta exempel 2, 3 och 11 att skäras på mitten.




Figur 8: Exempel där ett hörn är inuti objektet.Inspirerad av Paul Brouke [3].

Då alla hörn kan vara både i objektet eller utanför objektet har vi $2^8 = 256$ kombinationer av trianglar i kuben. Många av dessa är ekvivalenta på grund av spegling och rotation så det finns det ett antal unika kombinationer som kan ses i figur 9 [3].




Figur 9: Olika kombinationer av marching cubes, figur av Xu D och Zhang Y [6]. Figuren är lisensierad med creative commons (CC BY-NC-ND 4.0).

## 3.1 Förbättringsmöjligheter

Det finns ett antal olika sätt att förbättra resultatet för marching cubes. Nedan redogörs det för två enkla men effektiva metoder och även när de kombineras.

### 3.1.1 Mindre kuber

Ett enkelt sätt att göra hyfsade förbättringar till en viss nivå är att göra kuberna mindre [2].

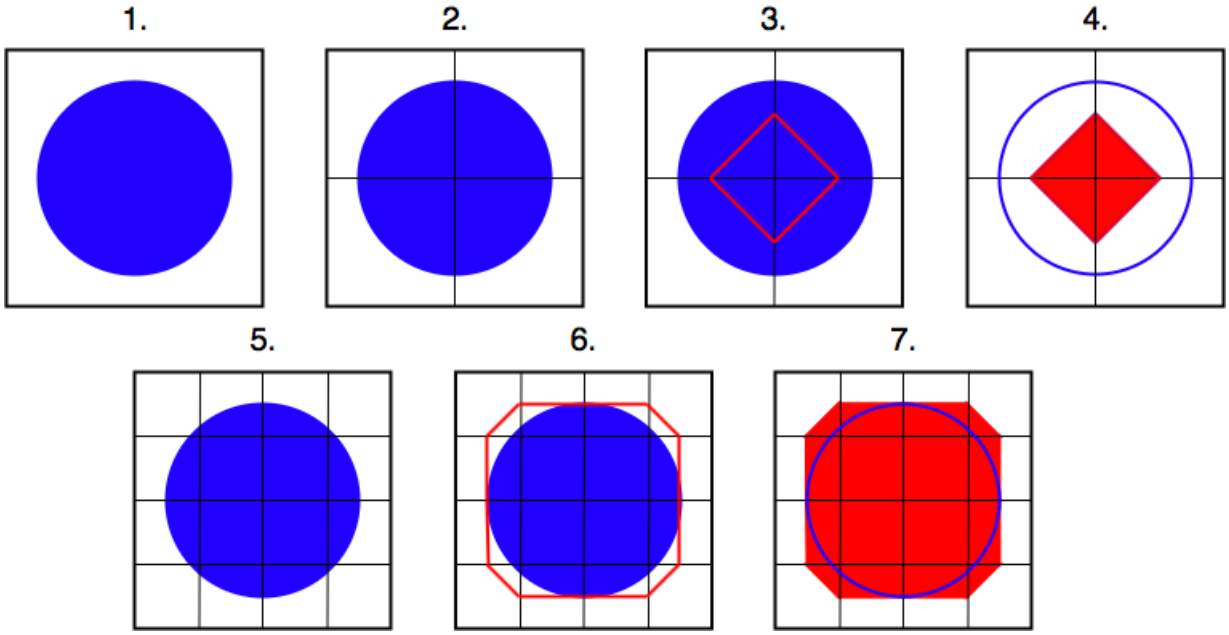
I figur 10 ser vi ett exempel på hur resultatet för bättras då kuberna minskar. I figuren är bilderna i 2D för enkelhetens skull men principen är den samma i 3D.

Nedan ges en kortfattad förklaring av stegen som utförs i figur 10.

1. Objektet och en yttre ram.

2. Dela in ramen i kvadrater.
3. Kontrollera vilket/vilka hörn som ligger i objektet och skapa en triangel för hörnet/hörnen.
4. Lägg ihop alla trianglar för att representera en yta för objektet.
5. Här delas ramen in i fler kvadrater.
6. Kontrollera vilket/vilka hörn som ligger i objektet och skapa en triangel för hörnet/hörnen.
7. Lägg ihop alla trianglar för att representera en yta för objektet.

I punkt 4 representeras ett ytobjekt men inte särskillt bra, i punkt 7 börjar det närma sig ett mer likt ytobjekt. Hade kvadraterna gjorts mindre hade såklart ytobjektet blivit mer likt objektet men som nämnts ovan fungerar det bara att minska rutnätet till en viss nivå sedan blir kostnaderna för beräkningarna för stora.




Figur 10: Exempel på konstruktion av ytor med stora och små kvadrater. Inspirerad av Ben Anderson [2]

### 3.1.2 Interpolation av kantlinjer

Interpolation är yttligare ett sätt att förbättra marching cubes algoritmen. Det hela bygger på att istället för att skära kantlinjen på mitten interpolerar man fram var kantlinjen skärs. Detta görs med hjälp av ekvation 8.

$$P = P_1 + \frac{(isovalue - V_1)(P_2 - P_1)}{V_2 - V_1} \tag{8}$$

Där $P_1$ och $P_2$ är hörnpunkterna för den kantlinje som skärs och $V_1$ och $V_2$ är de skalära värdena på respektive hörnpunkt. $P$ är punkten där kantlinjen skärs och $isovalue$ är isovärdet.

Åter till exemplet från avsnitt 3, figur 8. Nu interpoleras det fram var kantlinjerna skulle skäras om hörnen 0, 2, 3 och 7 skulle tilldelas slumpmässiga värden för respektive $V$ och ett värde för $isovalue$ mellan dessa $V$.

Hörn **0** = 5

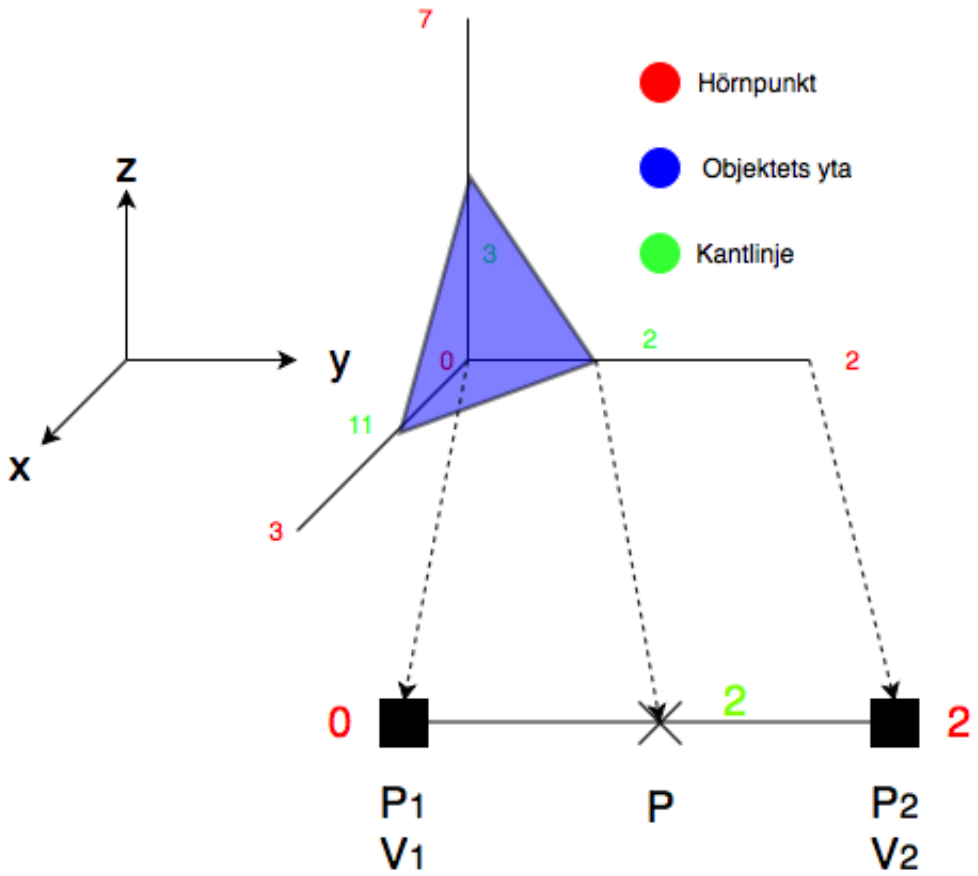
Hörn **2** = 3

Hörn **3** = 4

Hörn **7** = 1

Isovärdet **isovalue** = 2

I figur 11 ges ett förtydligande av beräkningen av $P$ i y-led.

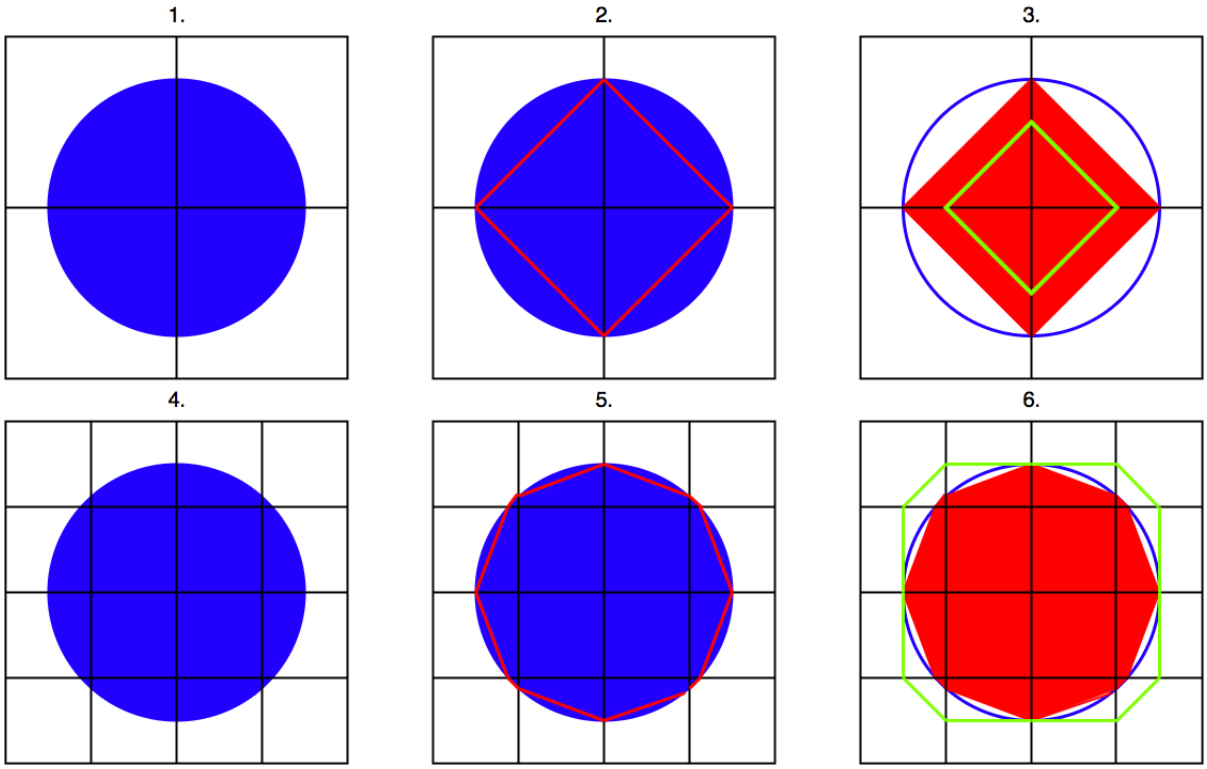



Figur 11: Exempel på interpolation av kantlinje i y-led.

- $P_1 = (0, 0, 0)$
- $P_2 = (0, 1, 0)$
- $V_1 = 5$
- $V_2 = 3$
- $isovalue = 2$

Sätts värdena ovan in i ekvation 8 skärs kantlinje 2 vid $P = (0, \frac{3}{2}, 0)$. På samma sätt beräknas det fram att kantlinje 3 skärs vid $P = (0, 0, \frac{4}{3})$ och kantlinje 11 vid $P = (3, 0, 0)$. Triangeln i figur 11 ska nu med hjälp av interpolering fått ett utseende som är mer likt den del av det faktiska objektet som finns i kuben.

Åter till 2D-exemplet från avsnitt 3.1.1 för att se hur interpolering faktiskt förbättrar resultatet. I figur 12, bild 3 ses hur interpolering ger yta som är mer lik objektet än utan interpolering. Den gröna inre linjekvadraten visar ytan utan interpolering. Kombineras interpolering och minskning av kvadraterna blir resultatet hyfsat nära originalet och en förbättring gentemot att bara minska av kvadraterna. Det gröna linjeobjektet visar ytan utan interpolering, se figur 12, bild 6.




Figur 12: Exempel på konstruktion av ytor med interpolation och kombination av interpolation och små kvadrater. Inspirerad av Ben Anderson [2]

# 4 Prestandamätning

Det här avsnittet presenterar mätningar gjorda i Inviwo för att jämföra prestandan hos volymsrendering med prestandan hos arching cubes.

## 4.1 Metod

Mätningarna utförs med hjälp av ett tidtagarur och manuell triggning av ombehandling av datan i Inviwo. Mätningarna utförs på en dator med följande specifikationer:

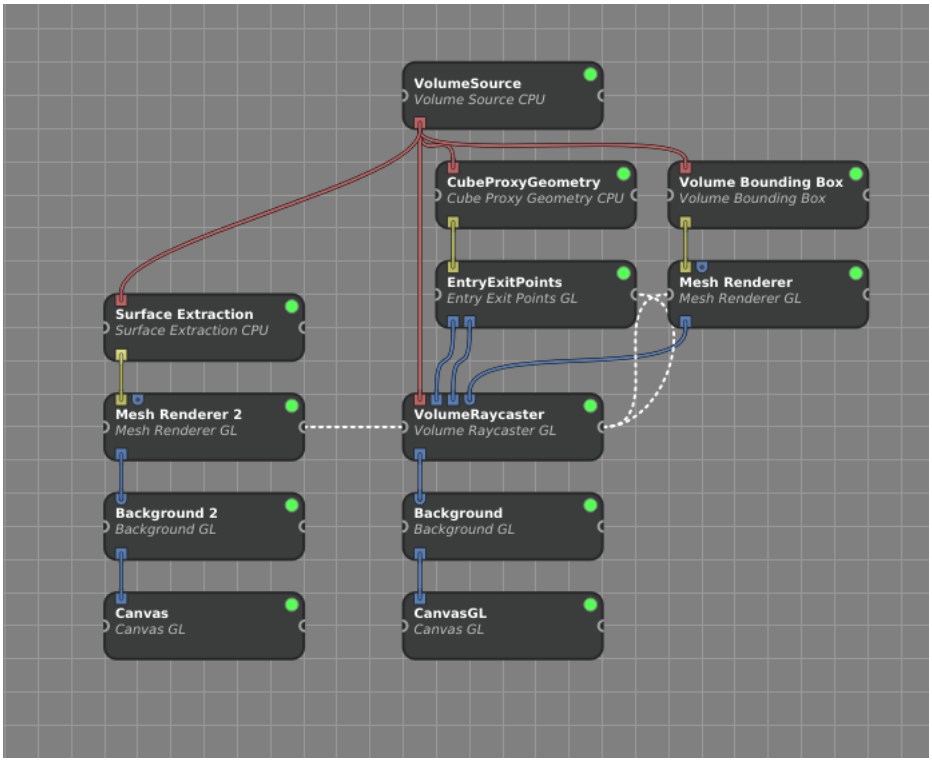
- Processor: Intel(R) Xeon(R) CPU E3-1225 v5 vid 3,30 GHz
- Grafikkort: NVIDIA GM107GL (Quadro K620) revision a2

- Primärminne: 16 GB DDR4

Versionen av Inviwo som används är v0.9.9.

### 4.1.1 Inviwonätverk

För att jämföra prestandan mellan marching cubes och volymsrendering används nätverket i figur 13. Den vänstra delen bestående av *Surface Extraction*, *Mesh Renderer 2*, *Background 2* och *Canvas* är ansvariga för att utföra marching cubes och rendera resultatet. *VolumeSource* läser in exempeldatan *boron.dat* som innehåller en volym på $150 \times 150 \times 150$ värden. Övriga processorer i nätverket är ansvariga för att utföra volymsrenderingen.




Figur 13: Inviwonätverk för prestandajämförelse.

Processorn *VolumeRaycaster* som är ansvarig för själva volymsrenderingen är inställd med följande inställningar:

- Classification: Transfer function
- Compositioning: Direct Volume Rendering
- Gradient computation: Central Differences

Dessa inställningar ger en direkt volymsrendering med en enkel färgkodad överföringsfunktion. Sampling rate, som påverkar hur många punkter på en stråle som skall samplas, ändras för att jämföra prestandan.

Processorn *Surface Extraction* som är ansvarig för att utföra marching tetrahedon, vilket är en variant av marching cubes men i grund och botten samma princip, får ett isovärde satt som varieras för att jämföra olika tider för olika värden [9].

Volymen som visualiseras är densamma för både volymsrenderingen och marching cubes. Den är en kub bestående av $150 \times 150 \times 150 = 3\ 375\ 000$ punkter.

## 4.2 Resultat

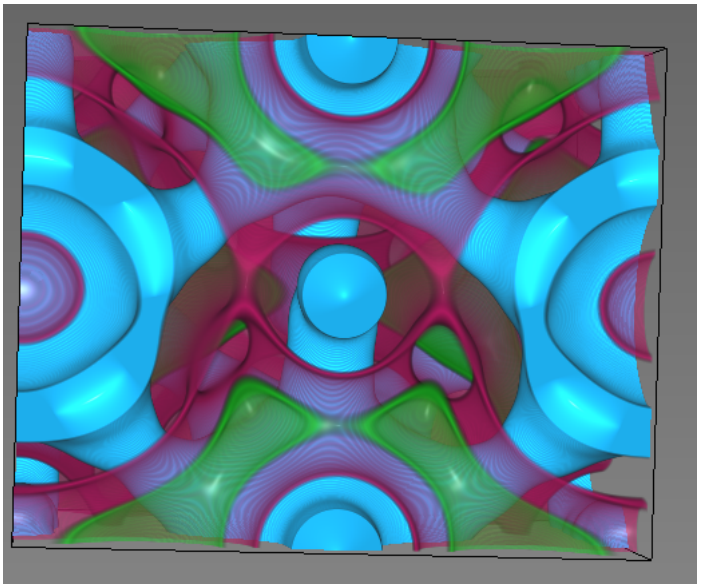
### 4.2.1 Volymsrendering

I tabell 4.2.1 finns resultaten från mätningarna gjorda för volymsrendering med olika antal sampel per stråle. I samtliga fall är kolumnen tid markerad med ett bindestreck då renderingen gick fortare än det var möjligt att manuellt starta och stoppa tidtagningen.

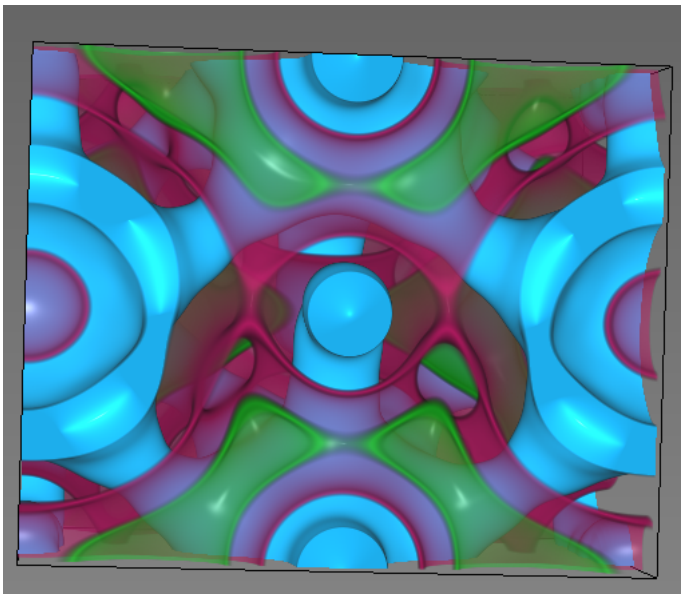
Tabell 1: Mätdata från tidtagning av volymsrendering.

| Sampling rate [sampel per stråle] | tid [s] |
|---|---|
| 1 | - |
| 8 | - |
| 10 | - |

Även om det inte gick att mäta med tidtagarur var det märkbart att det tog längre tid att rendera med högre antal sampel per stråle då programmet hackade om bilden roterades med 10 sampel per stråle, men inte med en sampel per stråle. Det var också en märkbar skillnad i kvalitet mellan bilden renderad med en sampel per stråle jämfört med bilden som gjordes med 10 sampel per stråle, se figur 14 och 15. Tydliga vågmönster kan ses i figur 14 som inte är närvarande i 15.



Figur 14: Volymsrendering med en sampel per stråle.



Figur 15: Volymsrendering med 10 sampel per stråle.

### 4.2.2 Marching tetrahedon

Marching tetrahedon tog betydligt längre tid att köras än volymsrenderingen på samma volym, vilket kan ses av resultaten i tabell 4.2.2.

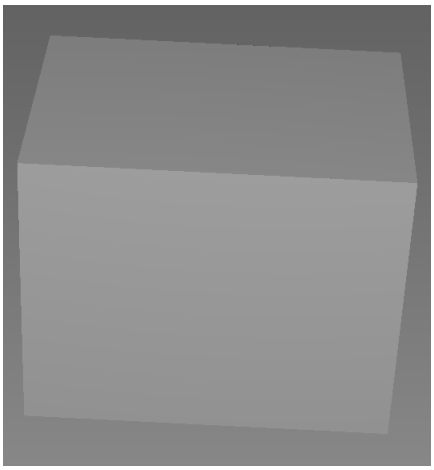
Tabell 2: Mätdata från tidtagning av Marching tetrahedon.

| isovärde | tid [s] |
|---|---|
| 0 | 13 |
| 1 000 | 14 |
| 10 000 | 25 |
| 20 000 | 20 |
| 40 000 | 6 |
| 65 000 | 4 |

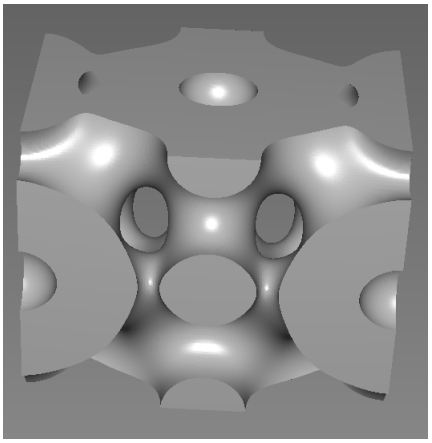
När algoritmen körs med isovärdet noll fylls hela volymen upp och ytan i figur 16 (a) är resultatet. Processen tar i det fallet 13 sekunder, enligt tabell 4.2.2. Då isovärdet skruvas upp stiger först beräkningstiden och detaljer framträder, men vid isovärdet 10 000 nås en brytpunkt och tiden sjunker igen. Under mätningarna var den längsta tiden vid ett isovärde på 10 000 med 25 sekunder och den kortaste tiden med ett isovärde på 65 000 som tog 4 sekunder att köra, se tabell 4.2.2.

Skillnaden i grafiskt resultat åskådliggörs i figur 16 (a-c). isovärdet noll får som synes hela volymen att fyllas, 10 000 skapar en mindre yta och 65 000 resulterar i nästan ingen yta alls.

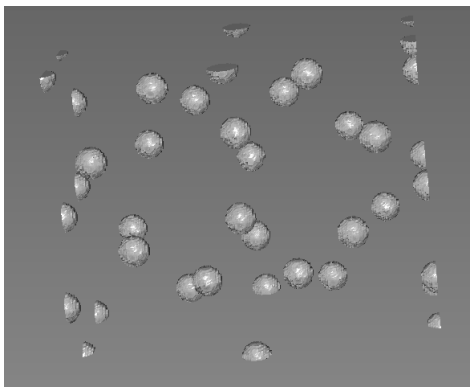
Till skillnad mot volymsrenderingen märks inget hackande i programmet när den konstruerade ytan roteras eller manipuleras på annat sätt. Någonting som är oberoende av vilket isovärde som valts. Marching tetrahedon-algoritmen och dess relativt långa körtid startar endast om indatan ändras eller någon inställning för algoritmen, t.ex. isovärdet, ändras.



(a) isovärde noll



(b) isovärde 10 000



(c) isovärde 65 000

Figur 16: Isoytor genererade med tre olika isovärden.

## 4.3 Diskussion

Den fösta, kanske mest uppenbara skillnaden, mellan volymsrendering och marching tetrahedon är tiden de tar att utföra. Volymsrenderingen går mycket fortare för en enskild bildruta, medan marching tetrahedon tar flera sekunder på sig att skapa en yta. I gengäld så går ytan skapad av marching tetrahedon mycket fortare att rendera när den väl är klar då algoritmen inte behöver köras om. Här kan en avvägning göras. Önskas t.ex. en rotation av ett väldigt stort objekt kan det hända att volymsrenderingen är för långsam och leder till långa väntetider mellan varje rotation, medan marching tetrahedon i det fallet tänkbart skulle kunna lämnas på över natten för att genera en yta som senare kan vridas i realtid.

Marching tetrahedon är i mätningarna betydligt långsammare än volymsrenderingen, men det är värt att notera att marching cubes eller marching tetrahedon måste gå igenom hela volymen. I det här fallet bestod volymen av 3 375 000 voxlar och en mindre volym skulle ge en betydligt kortare beräkningstid. Volymsrenderingen är inte lika starkt beroende av volymens storlek då den skickar in strålar i volymen och samplar på ett begränsat antal punkter.

Resultatet att marching tetrahedon först ökar i beräkningstid med ett ökat isovärde för att därefter gå fortare beror på att det tar längre tid att fylla ett volymselement än att lämna det tomt. Detta beror på att ett volymselement som inte är tomt måste, för marching cubes, matchas mot de olika möjligheter som ses i figur 9 och motsvarande för marching tetrahedon. Den beräkning som tog längst tid var för isovärdet 10 000, vilket skulle kunna förklaras av att den har många

kurviga ytor. När en yta är väldigt kurvigt leder det till att de mer komplexa fallen, de fall där många kantlinjer skärs, i figur 9 matchar och då måste mer interpolering utföras, vilket i sin tur ökar beräkningstiden.

Utöver prestandan kan det, från bilderna i resultatdelen, ses att volymsrenderingen och marching tetrahedron löser något olika problem. I projektet för elektronvisualisering behandlas bl.a. elektrontäthet i kristaller och då är det önskvärt med en bild som visar elektrontätheten genom hela kristallens volym. Detta gör att volymsrendering passar bättre. Ett annat fall är visualiseringen av något som kallas Fermi-ytor och som namnet antyder är det en yta som ritas utifrån en kristall. Datan för dessa Fermi-ytor är fortfarande en uppsättning voxlar i en volym, vilket medför att marching tetrahedon möjligtvis är att föredraga.

Både marching tetrahedon och volymsrendering har sina tillämpningsområden. Beroende på vad som skall visualiseras och den prestanda som önskas kan vilken som helst av de två väljas och de verkar generellt ge goda resultat.

# Referenser


[1] *5.11 CHG file*. 29 mars 1999. URL: http://cms.mpi.univie.ac.at/vasp/guide/node63.html#SECTION00071100000000000000 (hämtad 2018-04-25).

[2] Ben Anderson. *An Implementation of the Marching Cubes Algorithm*. URL: http://www.cs.carleton.edu/cs_comps/0405/shape/marching_cubes.html#1 (hämtad 2018-04-17).

[3] Paul Brouke. *Polygonising a scalar field*. URL: http://paulbourke.net/geometry/polygonise/ (hämtad 2018-04-17).

[4] *Computer Sweden. Uppslagninstabell*. URL: https://it-ord.idg.se/ord/uppslagningstabell/ (hämtad 2018-03-19).

[5] *Computer Sweden. Volumetrisk*. URL: https://it-ord.idg.se/ord/volumetrisk/ (hämtad 2018-03-19).

[6] Xu D och Zhang Y. *Generating triangulated macromolecular surfaces by euclidean distance transform*. 2009. URL: https://zhanglab.ccmb.med.umich.edu/papers/2009_3.pdf (hämtad 2018-05-23).

[7] Matthew Fisher. *Marching Cubes*. URL: https://graphics.stanford.edu/~mdfisher/MarchingCubes.html (hämtad 2018-04-17).

[8] James D. Foley m. fl. *Computer Graphics: Principles and Practice*. Addison-Wesley, 1990. ISBN: 0201121107.

[9] Charles D. Hansen och Chris R. Johnson. *Visualization Handbook*. Academic Press, 2004. ISBN: 012387582X.

[10] Milan Ikits m. fl. *GPU GEMS. Chapter 39.2*. URL: https://developer.nvidia.com/gpugems/GPUGems/gpugems_ch39.html (hämtad 2018-03-09).

[11] Milan Ikits m. fl. *GPU GEMS. Chapter 39*. URL: https://developer.nvidia.com/gpugems/GPUGems/gpugems_ch39.html (hämtad 2018-03-09).

[12] *Inviwo. Features*. URL: http://www.inviwo.org/user-documentation/features/ (hämtad 2018-03-09).

[13] Barthold Lichtenbelt och Randy Crane. *Introduction to Volume Rendering (Hewlett-Packard Professional Books)*. Prentice Hall, 1998. ISBN: 0138616833.

[14] Jörgen Malmquist, Staffan Cederblom och Susanne Wikman. *Nationalencyklopedin. magnetisk resonanstomografi*. URL: http://www.ne.se/uppslagsverk/encyklopedi/l%C3%A5ng/magnetisk-resonanstomografi (hämtad 2018-04-25).

[15] *Mathworks. Overforingsfunktion*. URL: https://se.mathworks.com/discovery/transfer-function.html (hämtad 2018-03-19).

[16] *Nationalencyklopedin. gradient*. URL: https://www.ne.se/uppslagsverk/encyklopedi/l%C3%A5ng/gradient (hämtad 2018-03-19).

[17] *Nationalencyklopedin. implicit funktion*. URL: https://www.ne.se/uppslagsverk/encyklopedi/l%C3%A5ng/implicit-funktion (hämtad 2018-04-25).

[18] *Nationalencyklopedin. interpolation*. URL: https://www.ne.se/uppslagsverk/encyklopedi/l%C3%A5ng/interpolation (hämtad 2018-04-25).

[19] *Nationalencyklopedin. sampling*. URL: http://www.ne.se/uppslagsverk/encyklopedi/l%C3%A5ng/sampling (hämtad 2018-04-26).

[20] *Nationalencyklopedin. Tetraeder*. URL: https://svenska.se/tre/?sok=tetraeder&pz=2 (hämtad 2018-04-27).

[21] Richard C. Penney. *Linear algebra. ideas and applications.* Hoboken, New Jersey : Wiley, 2016., 2016. ISBN: 9781118909584.

[22] Anders Ramgard. *Vektoranalys*. Teknisk högskolelitteratur i Stockholm AB, 2000.

[23] *SciVis Group. Inviwo*. URL: http://scivis.itn.liu.se/research/inviwo/ (hämtad 2018-03-09).

[24] *Svensk ordbok. skalär*. URL: https://svenska.se/tre/?sok=skal%C3%A4r&pz=1 (hämtad 2018-04-25).