



## **HUR KONTURRENDERING FÖR SPEL ANVÄNDS INOM AKADEMIN OCH SPELINDUSTRIN**

En litteraturöversikt över konturtekniker

## **HOW OUTLINE RENDERING FOR GAMES IS USED IN ACADEMIA AND THE GAME INDUSTRY**

A Literature review of outline techniques

Examensarbete inom huvudområdet  
Informationsteknologi  
Grundnivå 15 högskolepoäng  
Vårtermin 2026

Robin Sjölin

Handledare: Mikael Johannesson  
Examinator: Mikael Thieme



# Sammanfattning

Konturrendering är en teknik som kommer i många former med sina egna för- och nackdelar. Hur fungerar dessa olika metoder och varför väljs vissa metoder över andra? Detta är en litteraturöversikt som kommer analysera litteraturen inom tekniker för att producera konturer inom 3D grafik för realtidsapplicationer med fokus på spel. Målet med denna översikt är att ge läsaren en klar bild över hur dessa tekniker används och modifieras för att passa utvecklarens önskad estetisk effekt. Här uttrycks implementeringsdetaljer och evalueringsfrågor som kan påverka valet av implementeringsmetod. Här analyseras de följande metoder: Suggestive contours, Canny edges, Sobelfilter, Laplacianfilter och Inverse hull. Analysen kommer fram till att många implementeringsmetoder följer stegen av att använda någon av dessa konturalgoritmer och sedan modifiera och kombinera linjer skapat av dessa algoritmer för att uppnå den önskade estetiken ofta via vertex shaders.

**Nyckelord:** Icke-fotorealistisk rendering, konturkonturrendering, Datorgrafik, Grafikprogrammering, Stiliserad grafik

# Innehållsförteckning

<b>1</b>	<b>Introduktion .....</b>	<b>I</b>
<b>2</b>	<b>Bakgrund .....</b>	<b>II</b>
<b>3</b>	<b>Problemformulering.....</b>	<b>V</b>
3.1	Metodbeskrivning.....	V
<b>4</b>	<b>Analys .....</b>	<b>VIII</b>
4.1	Historiska implementationer .....	VIII
4.2	Moderna implementationer.....	XII
4.3	Stiliseringstekniker .....	XV
4.4	Mönsteranalys.....	XVI
<b>5</b>	<b>Sammanfattning och diskussion .....</b>	<b>XIX</b>
5.1	Sammanfattning.....	XIX
5.2	Diskussion .....	XX
5.3	Samhälleliga och etiska aspekter .....	XXII
5.4	Framtida arbete.....	XXII
	<b>Referenser.....</b>	<b>XXIII</b>

# 1 Introduktion

konturrendering är en teknik för att skapa linjer runt objekt för att emulera den estetik som ses inom viss 2D-konst. Tekniken bygger på icke-fotorealstisk rendering som är en renderingsmetodologi som inte siktar på att skapa grafik som är så realistisk som möjligt. Denna metodologin fokuserar på läsbarhet och stilisering och används inom exempelvis medicin, designindustrier och spel. Konturer används för att definiera och separera objekt. I 3D är detta betydligt mer avancerat än i 2D-konst eftersom konturer måste beräknas utifrån geometrin och kameran, vilket skapa tekniska utmaningar där 2D-konst får det nästan gratis via skissar.

Inom spel används icke-fotorealstisk rendering och konturer för både estetik och funktion. I spel som Borderlands används konturrendering och har blivit en grundläggande del av spelseriens identitet. Icke-fotorealstisk rendering har visat sig att kunna förbättra spelarens upplevelse och immersion. Spel sätter särskilda krav eftersom rendering måste fungera i realtid och från alla kameravinklar. Detta skapar begränsningar på vilka tekniker som kan användas och kan göra implementationen mer komplex. Valet av konturteknik behövs därför göras tidigt i utvecklingsprocessen för att passa både tekniska krav och visuell stil.

Denna studie genomför en litteraturoversikt baserad på akademiska databaser samt industrikällor. Fokus ligger på 3D konturrendering, medan 2D-tekniker exkluderas. Artiklar analyseras utifrån implementation, användning och begränsningar. Målet är att utveckla en förståelse över teknikerna för att formulera idéer om vilka tekniker som fungerar för vissa spel.

I moderna implementationer av konturrendering dominerar screen-space metoder som Sobel- och Laplacianfilter eftersom de är snabba, flexibla och fungerar väl i realtidsmotorer. Dessa kombineras med olika datakällor och efterbearbetning för att uppnå den önskade stilen. Geomtribaserade metoder används även för att producera konturer där stabilitet är en viktig faktor, dessa metoder är dock mindre flexibel eftersom det blir knepigt med halvtransparens och kontureffekter. I med att datorer alltid blir snabbare har flera av dessa tekniker kombinerats och bearbetats för att uppnå en balans mellan prestanda, visuell kvalitet och stilisering.

## 2 Bakgrund

Icke-fotorealistisk grafik är ett område inom datorgrafik som fokuserar på att skapa stiliserade visuella uttryck snarare än att efterlikna verkligheten. Till skillnad från fotorealistisk rendering, där målet är att simulera ljus och material på ett fysiskt korrekt sätt. Denna typ av grafik fokuserar på ofta på stilisering och tydlighet snarare än realism. Denna grafik används inom spel, animation och visualisering. Icke-fotorealistisk rendering prioriterar tydlighet, formdefinition och visuell kommunikation. Denna grafik används inom flera områden, såsom spel, animation, teknisk visualisering och utbildningsmaterial.

Cole, F. et al. (2008) påpekar att konturer framkommer mest inom 2D-konst såsom illustrationer och tekniska ritningar. Speciellt inom tekniska ritning har konturer som fördel att definiera former, separera objekt och stilisering. Dessa implementeras i 3D för att uppnå liknande kriterier, skillnaden med 2D och 3D är att i 2D kommer det med mycket enklare grund för att producera konturer. Detta gör att det uppstår nya tekniska utmaningar då konturer inte längre är explicit definierade utan måste beräknas utifrån geometrin och kameran.

Enligt G. Winkenbach och D. Salesin (1994) är den visuella representationen inom medicinska dokument ofta handritad eftersom det tillåter små och ofta gömda strukturer att vara mycket bättre beskrivet. Inom Boeing är manualer visuella element handritade även om de har dessa finns i CAD databaser eftersom handritade versioner är ett mer effektivt kommunikationsverktyg än vad som kan produceras av datorn. Detta är något som var ett klart problem för den tiden och i mer modern tid finns det betydligt mer forskning inom området som har ändrat hur vi producerar visualiseringar.

Inom spel finns det flera exempel på inflytelserika spel som använder sig av denna typ av grafik för att stödja både estetisk stilisering och funktionella krav som läsbarhet. Spelserien Borderlands är ett exempel på en inflytelserik spelserie som populariserade konturrendering bland AAA titlar och konturrenderingen bland deras spel är något som är starkt förknippat med spelets identitet och därmed även dess estetisk stilisering. Spel där konturrendering används har det ofta som något som ger spelet sin identitet och kan använda det för att relatera till andra verk. Spel som Dispatch använder konturrendering för att skapa ett starkt samband mellan deras spel och superhjältar från serietidningar. I figur 1 nedan visas en bild från Borderlands som exemplifierar den stiliserade grafiken i Borderlands-serien.



**Figur 1.** Borderlands 2 från Steam (u.å.)

Lake, A. et al. (2000) tyder också att icke-fotorealistisk rendering kan bredda förmågan att kommunicera tankar och känslor genom datorer och att denna typ av rendering gör det möjligt att publiken har större immersion av en upplevelse. Detta kan tyda på att spelaren behandlar den information i icke-fotorealistisk rendering annorlunda än fotorealistisk rendering. Spelet *Return of the Obra Dinn* berättar en dystert berättelse vilket grafiken lyfter genom att begränsa mängden färger och visuell information som kan ses i följande figur 2.



**Figur 2.** Return of the Obra Dinn från Steam (u.å.)

Spel skiljer sig från många andra tillämpningsområden genom att renderingen måste vara robust ur alla kameravinklar och fungera i realtid. Eftersom spel äger plats i en interaktiv miljö och spelaren fritt kan röra kameran, måste kontureffekten vara stabil oavsett vy. Detta står i kontrast till exempelvis tekniska illustrationer, där bilden ofta är statisk och kan korrigeras manuellt i efterhand. Detta gör att vissa tekniker som används inom vissa andra områden inte

är möjligt inom spel. Detta betyder att algoritmer såsom enkla filter som behöver justeringar för att det ska fungera konsekvent är opassande för implementation i spel.

I de flesta realtidsapplicationer finns två typer av shaders som kan modifieras för att producera vissa effekter. Användning av GPU:n och följandet av dess begränsningar är viktigt för att en algoritm ska fungera i realtid. GPU:n har inte tillgång till samma data som CPU:n har och gör allt parallellt vilket gör att beräkning av ny data som GPU:n räknar parallellt inte är möjligt.

Vertex shaders och fragment shaders. Enligt Vries (2020) är vertex shaders en typ av shader som borde ta in information från hörnpunkten av ett objekt vilket kan definieras av utvecklaren. Detta kan modifiera egenskaper och data hos hörnpunkten och skapa information som skickas till fragment shadern. En fragment shader är en shader som händer per pixel för en yta på ett objekt där data interpoleras från data av vertex shadern. Detta är en vanlig men också potentiellt begränsande eftersom all grafisk beräkning måste nästan alltid utgå från dessa shaders.

Shaders har också tillgång till extern data, vanligtvis texturer som kommer från kameran. Enligt Unity (2017) kan depth maps och normal map hämtas från kameran i deras spelmotor. Dessa texturer kan hämtas via vanlig kod eller shaders. Normal texturen är en textur som har view-space normaler vilket betyder att texturen har värden som pekar vart normalen pekar mot som baseras från vart kameran är. Unity (2017) tar fram att en depth texture är en texture som visar djupet av vad kameran ser.

Termer som beskriver exakt hur linjer ritas ut på ett objekt är inte speciellt fastslaget och kan eventuellt användas löst. Men generellt inom området används termerna som silhuett och kontur för att syfta på de linjer som ritas längs kanter. Silhuett tyder oftare dock på linjerna som ritas runt objekt utan några linjer innanför objektet. Konturer syftar ofta på silhuettlinjerna och linjer ritade innanför objektet.

Eftersom mycket av beräkningarna görs på hörnpunkt eller pixel kan många grupperas inom vissa kategorier av konturalgoritmer. De två största områden för detta blir därför screen-space och world-space. Det finns även hybrid algoritmer som använder en kombination av beräkning på pixel och hörnpunkt.

Det kan vara en stor utmaning för utvecklare att besluta sig över hur de ska implementera konturrendering. Detta är ett val som måste göras tidigt i arbetsprocessen och de flesta fall måste tidigt implementeras. Detta är för att tillåta 3D konstnärerna att producera modeller som matchar väl med den stil som kommer med konturrendering. I mer modern litteratur kring konturrendering sätts dessa ihop med större icke-fotorealistiska renderingssystem vilket kan sakta ner beslutningstiden för att söka det system som utvecklaren är efter.

Det finns flera exempel på spel som använder dessa tekniker. Inom icke-fotorealistiskt rendering är konturrendering en teknik som medelstor inom området, det finns flertal icke-fotorealistiskt renderade spel som inte använder tekniken och flera som gör det. Hur dessa tekniker appliceras kan även variera mycket eftersom det finns mycket rum för exakt hur konturer ska se ut. Konturer kan vara en subtil effekt som finns på enstaka som i Genshin Impact eller kan det vara en essentiell del av spelets grafik som i Ökami. Användning av effekten är fortfarande populär men är mindre i mer moderna spel och vanligtvis är en subtil effekt.

## 3 Problemformulering

Konturrendering inom 3D-grafik är vanligt använt inom icke-fotorealistisk rendering för att stärka formuppfattning, visuell tydlighet och stilistiskt uttryck. Det finns flera metoder för att uppnå konturrendering i 3D som screen-space shaders, vertex shaders och stencil testing. Dessa har använts utförligt och varje metod har olika egenskaper och kommer med olika beräkningskostnader, anpassningsmöjligheter och visuella egenskaper. Historiskt sett har konturer varit starkt förknippade med tecknade serier och illustrativa stilar, vilket speglar konventioner från 2D-konst.

Dock är det vanligt att detta undviks i icke-fotorealistisk 3D-grafik, särskilt i nya spel. Det är vanligast att se spel som undviker konturrendering i dess visuella stil. I stället föredrar stilar som närmar sig platta färger, målningar eller realism-inspirerade stilar. Detta väcker frågor om relevansen och effektiviteten av konturrendering. Trots teoretiska fördelar för visuell läsbarhet verkar konturrendering användas mindre ofta i praktiken, vilket kan tyda på eventuella begränsningar i estetik, prestanda eller produktionsflöde.

Problemet denna litteraturöversikt behandlar är bristen på samlad och fokuserad uppfattning för hur tekniker för specifikt konturrendering i moderna sammanhang och i vilka situationer de ger meningsfulla fördelar. Befintlig forskning fokuserar främst på konturrendering i större renderingssystem medan färre studier granskar, konstnärliga begränsningar på flera konturtekniker. Denna litteraturöversikt syftar därför till att sammanfatta tekniska och praktiska perspektiv för att identifiera styrkor och brister hos konturrendering.

### 3.1 Metodbeskrivning

Denna studie använder en litteraturöversikt för att analysera befintlig forskning och industrikällor om konturrendering inom 3D-grafik. Målet med denna översikt är att förstå hur konturtekniker fungerar, deras fördelar och begränsningar för att underlätta beslutandet av hur olika tekniker påverkar grafiken.

För detta ställs frågeställningen: *Hur skiljer sig olika tekniker för konturrendering i 3D-grafik åt inom implementation, prestanda och visuella egenskaper i spel?* Med denna frågeställning utforskares detaljer som skapar en bra bild på vad beslut som ska ta baserat på utvecklarens mål och behov. Denna översikt kommer även att analysera litteraturen om det finns luckor i olika typer av källor. Detta tar även upp en sekundär frågeställning: *Vilka luckor finns i forskningskällor jämfört med industrikällor?*

Litteraturöversikten kommer att söka efter litteratur primärt på Högskolan I Skövdes LibSearch som samlar flera databaser. Primärt kommer fokus ligga på databaser som är relevanta för datorgrafik som ACM Digital Library och IEEE Xplore. Industrikällor kommer också att användas, dessa kommer hittas genom att söka igenom databaser som GDC Vault och sökande av direktkällor från till exempel Unity Unite. Ytterligare källor kan även hittas i referenser från källor som kan leda till mer grundläggande forskning inom området.

Artiklar som fokuserar på konturalgoritmer i 2D eller andra stiliseringsmetoder kommer uteslutas eftersom det ger inga insikter i de tekniska aspekterna av 3D konturrendering. Även om trender och estetiska rörelser är en styrande faktor till den moderna användning av konturrendering kommer denna text inte fokusera på de aspekter and sätter inte målet att ge

en helhetsbild på den aspekten.

Artiklar som kommer analyseras kommer vara artiklar som tar upp teknisk implementation, användning och begränsningar. Dessa ämnen kommer jämföras och syntetiseras för att förstå den tekniska grunden till tekniken, effekten den har och hur den kan användas för att förstå effektens påverkan och effekt. Artiklar kommer grupperas inom renderingsmetod och visuella egenskaper.

Denna analys syftar på att samla information och först bygga en förståelse för effektens teknisk grund. Detta kommer att stödja det sekundära målet med denna artikel och det är att förstå effektens effekter på de tekniska och. Återkommande metoder, egenskaper och begränsningar kommer identifieras och syntetiseras. Aspekter som beräkningskostnad, visuella egenskaper och effekter kommer att jämföras för att skapa en sammanhängande förståelse för hur konturrendering fungerar och var de brister.

Metoder för konturrendering kommer att evalueras inom vissa teman såsom objektseparation, djupuppfattning, visuell betoning och möjlighet för stilisering. Dessa metoder kommer jämföras för att forma en uppfattning över varför en metod skulle väljas än det andra inom det evaluerande området.

Denna strukturerade jämförelse stöder utvärderingen av konturrendering både som en teknisk lösning och ett visuellt verktyg. Syntesen används vidare för hjälpa spelutvecklare att tolka hur tekniska begränsningar och perceptuella resultat kan bidra till deras spel.

För att analysera luckor används metoden av Backlund & Hendrix (2013) användas. Detta ger oss en bild på luckor som kan finnas för både akademiska och industriella källor. Metoden de skapar och kommer användas här följer dessa steg:

- Definiera omfattning
- Systematiskt söka databaser
- Besluta på urval
- Granska urval för att exkludera opassande artiklar
- Skapa syntes baserat på artiklarna

Omfattandet av denna analys kommer att inkludera som fokuserar på hur konturrendering används och implementeras. Det kommer inte sättas någon gräns på tidsram eftersom detta är ett område som har relativt mindre artiklar och många tekniker som används idag är tekniker skapade för länge sen. Konturrendering som används utanför generell datorgrafik eller spelgrafik kommer att exkluderas som till exempel datorseende eller neurovetenskap. Artiklar som fokusera på relevant område men inte specifikt på konturrendering kommer också såklart exkluderas. För denna analys kommer HIS LibSearch att användas med dessa sökord:

- "outline rendering" OR "edge rendering" OR "silhouette rendering"
- ("non-photorealistic rendering" OR NPR) AND outline

Industrikällor kommer sökas i GDC Vault, Google, 80 Level och Game Developer. Eftersom sökningsverktyg är simplare på flera av dessa källor och har mer irrelevant information på dessa resurser kommer sökstrategierna behöva vara annorlunda. Sökningar kommer vara

delar av de tidigare sökord. Begränsningar kommer behövas göras i mängd källor på grund av dessa sidors icke-strikta sökningsverktyg. Diskussioner som forumfrågor kommer att exkluderas. Det finns även möjligheter att flera artiklar täcker samma metod, om detta hittas exkluderas en av dem artiklarna.

Under granskningen kommer artiklarna att kategoriseras inom flera kategorier. Författare inkluderas först för att hänvisa källan som används. År kommer också inkluderas för att hänvisa källan och för att kanske se om något mönster dyker upp. Ursprung kommer syfta på om källan är akademisk eller industriell. Renderingstyp syftar på vad för typ av rendering som tas upp i artikeln. Typ syftar på meningen och vad som är relevant för den texten, *innovation* tyder på ny innovation inom områden, *modifiering* tyder på nya användningar a redan existerande tekniker och *översikt* tyder på text som inte ger någon ny information. Realtid syftar på om tekniken kan användas i realtid på dagens datorer.

Som en litteraturbaserad studie är denna översikt begränsad av tillgängligheten och inriktning på befintlig forskning. Analysen bygger på hur författarna formulerar och utvärderar konturtekniker vilket kan leda till att informell produktionskunskap fattas. Trots begränsningarna kring trender och informell information ger denna artikel en omfattande förståelse kring grunden för tekniken inom hur den fungerar och dess effekt.

## 4 Analys

Först diskuteras metoder för att producera konturer för att besvara den första frågeställningen. Detta ska skapa en bra inblick över hur konturmetoder fungerar och deras egenskaper. Det finns flera metoder för att implementera kontureffekter, flera som fungerar för spel och flera som inte fungerar. I denna del kommer implementationer som inte passar spel för att formulera en förståelse över varför vissa metoder inte fungerar.

Efter de historiska implementationerna tar implementationer som används i modern spelgrafik. Dessa blir relevant för att lära exakt hur metoder implementeras för att skapa en konkret förståelse för att senare diskutera prestanda och visuella egenskaper.

Efter detta kollas på flera källor enligt metoden av Backlund & Hendrix (2013) för att formulera en idé över vilka luckor som finns inom industriella- och forskningskällor. Detta kommer senare att diskuteras för att förstå vad resultaten av detta betyder.

### 4.1 Historiska implementationer

Mycket av dagens konturrenderingssystem bygger på eller baseras på äldre system. Dessa algoritmer är vanligtvis inte direkt använt i moderna spel men skapar kontext och grund till de moderna konturrenderingsteknikerna idag. Dessa algoritmer kan inte direkt användas i spel på grund av begränsningar i moderna realtidsrenderingspipelines.

DeCarlo et al. (2003) definierar konturer som  $n(p) \cdot v(p) = 0$  där  $n$  är normalvektorn av ytan,  $v$  är vyvektorn från kameran och  $p$  är en punkt på objektets yta. Detta syftar att ytor där  $n(p) \cdot v(p) > 0$  riktas mot spelaren och kameran är  $n(p) \cdot v(p) < 0$  pekar bort från kameran. Denna artikel tar upp hur detta fungerar i generell rendering men relevant för spel och ofta generell modern rendering är att eftersom det byggs av trianglar är att systemet måste upptäcka vart denna övergång mellan positivt och negativt händer eftersom det är högt ovanligt att en triangel är exakt parallell jämt kamerans vyvektor. Detta behövs göras genom att analysera granntrianglar för att upptäcka där denna beräkning går från positivt till negativt.

Denna artikel bygger på modeller för att bygga konturer. En geometri-baserad approach och en Screen-space approach båda använder mycket av samma underliggande definitioner men screen-space algoritmen approximerar mycket mer av informationen.

Detta är grunden till deras kontursystem men de påpekar även att detta missar också annan viktig information som gropar, utbuktningar och veck. Detta gör att dom introducerar suggestiva konturer som är konturer som bör komma upp om kameran flyttar på sig lite. Dessa ska ligga inom objektet som renderas som är annorlunda till vanliga sillouetter. Detta är menat att ge objektet en mer definierad form och visa saker som gropar, utbuktningar och veck.

Medan konturvillkoren  $n(p) \cdot v(p) = 0$  definierar silhuetter väl, påstår DeCarlo (2003) att silhuetter enbart inte är tillräckliga för att skapa detaljerade former. Silhuetter anger endas var ytan blir vinkelrät mot vyvinkeln. De visar inte mycket av de interna ytor som utbuktning, gropar eller veck som ger viktig information. För att åtgärda detta introducerar dom suggestiva konturer vilket kan beskrivas som konturer som förekommer när vyn ändras lite. Dessa konturer ritas på den visade delen av ytan för att bättre kommunicera lokal formvariation utan att bero på skuggning.

Suggestiva konturer kan definieras som punkter där dessa formler stämmer  $k_r(p) = 0$  och  $D_w k_r(p) > 0$ .  $k_r(p)$  är värdet på kurvan och  $D_w k_r(p)$  syftar på en andra ordningens partiella derivata av  $k_r(p)$ . Detta är tillräckligt för att producera en bild av ett objekt som har konturer men det kan stöta på märkbara problem när det kommer till animation vilket blir speciellt problematiskt i spel. Detta är eftersom algoritmen med den nuvarande information är instabil vilket betyder att suggestiva konturer kan plötsligt dyka upp, försvinna plötsligt och hoppa plötsligt. Detta händer huvudsakligen från radiella riktningen som kommer när den radiella kurvan räknas ut. När normalen och vyvektorn är lika produceras en extrem liten vektor vilket resulterar i att radiella kurvan blir instabil eftersom små ändringar orsakar stora ändringar av vinkeln på en vektor. Författarna tar upp ett tillstånd som upprätthåller ett tröskelvärde, detta är  $0 < \theta_c < \cos^{-1} \frac{n(p) \cdot v(p)}{\|v(p)\|}$ .  $\theta_c$  är ett värde satt av utvecklaren för att uppnå den stabilitet som de vill ha, författarna rapporterar att de använder värden mellan 20 och 30 grader för  $\theta_c$ . Detta tillstånd gör att värdet mellan normalen och vyvektorn är större än  $\theta_c$  vilket undviker ostabila tillstånd.

Denna konturimplementation använder sig av flera koncept som ofta är problematiskt i användning i realtidsrendering. Kurvaturer, riktningar, radiella kurvor och riktningderivata är ofta orealistiskt att använda i spelmotorers grafikpipeline. Dessa faktorer i realtidsmotorer går att approximera men kan bli problematiskt med prestandan, speciellt när det kommer till meshes med skinning eftersom förberäkande faktorer måste räknas om vid rörelse. Detta gör att realtidsapplicationer använder sig främst av deras bild-baserad implementation som grundar sig i samma logik som den geometribaserade algoritmen som nyss togs upp. Denna implementation använder sig av information som ofta genereras vid normal realtidsrendering, detta gör det till något som ofta är mer relevant för spelrendering. Detta gör att det även passar väl ihop med redan existerande renderingspipelines.

Bild-baserade implementationen baseras på ett pass som producerar svartvit bild av scenen som produceras via  $\frac{n \cdot v}{\|v\|}$ , det enda datan som behövs från scenen via denna funktion är normalen av ytan och vyvektorn från kameran. För varje pixel hämtas de intilliggande pixlarna i ett cirkulärt område, i detta område är  $p_i$  den pixel som nu kollas på och  $p_{max}$  den ljusaste pixel i det cirkulära området. Algoritmen kollar sen på om den möter två tillstånd. Det första är om  $p_i$  är mörkare än en viss andel av områdets pixlar som kontrolleras av  $s$ , där om  $s = 0.1$  måste  $p_i$  vara mörkare än 90% av alla pixlar i området. Det andra tillståndet är  $p_{max} - p_i > d$  där  $d$  är fast värde som kontrolleras av användaren, detta tillstånd säkerställer att pixel är tillräckligt brant för att bli en kontur. Tillsist appliceras ett medianfilter av radien  $r$  för att jämna ut resultatet och ta bort brus.

Både object-baserade systemet och bild-baserade systemet har sina för- och nackdelar. Det objekt-baserade systemet ger oss själva geometrin som ger oss tillgång till mycket mer avancerad användning som varierad tjocklek men kommer med mycket tyngre matte för att räkna ut dess geometri. Det största problemet med det objekt-baserade systemet i realtidssystem är dock hur dåligt det integrerar med redan existerande renderingsystem och meshes med skinning. För att producera object-baserade konturer behövs flera rendering passes genomföras som ofta är beräkningsmässigt kostsamma. Detta gör att lösning ofta inte passar i realtidsrendering även med de snabbare datorer som finns idag. Den bild-baserade implementationen passar mycket bättre ihop med moderna renderingsarchtekturer. Algoritmen är mycket snabbare och mer robust eftersom det kan utjämnas enklare men stöter

på problem i att det är en approximation och inte en exakt definition.

En till historisk artikel som implementerar en metod för konturrendering är Canny (1986) edges. Denna artikel fokuserar på matematiska optimala sätt att extrahera konturer från bilder. Detta edge-detection system är designad till att uppfylla tre kriterier. Systemet ska detektera riktiga kanter och undvika noise, det ska noggrant lokalisera kanter vilket betyder att det ska producera en kant exakt vart kanten faktiskt är och till sist ska systemet svara endast en gång på varje kant, det ska inte finnas flera linjer vid kanter.

Canny (1986) algoritmen börjar med att applicera ett Gaussian blur steg som gör bilden suddig vilket görs för att ta bort noise innan edge detection. Detta skapar vissa kompromissar där en större oskärpa leder till bättre brusreducering men mindre oskärpa ger bättre precision. Gaussian blur görs där  $\sigma$  kontrollerar storleken av oskärpan. Detta räknas på själva bilden via en där  $I$  representerar ljusstyrkan av den inmatade bilden.

Med den suddiga versionen av inmatningsbilden kollas den för stora skillnader i ljusstyrka. I denna text görs detta via derivator. Detta bildar ett rutnät av vektorer som pekar mot den största ökningen med den magnituden som matchar förändringarna

Nu går algoritmen igenom kanttunnande eftersom utan detta steg blir kanterna tjocka. Pixlarna som går kors från vinkeln av derivata som kollas för om den är den tjockaste i den linjen. Detta görs genom att kolla på magnituden av derivata på de pixlarna längs den linjen. Om pixeln med det största värdet är den nuvarande pixeln behålls den pixel och med inte är det sätts pixels värde till noll. Detta hjälper att hålla linjetjockleken enhetligt.

Vid detta steg klassificeras pixlarna baserat på den modifierade magnituden. Det finns två tröskelvärden som kontrollerar detta, ett lågt och högt tröskelvärde,  $T_L$  och  $T_H$ . Om värdet är över det höga tröskelvärde markeras det som en stark kant, om det är mellan både tröskelvärde är det en svag kant och till sist om det är under lägre tröskelvärde är det brus som turs bort. Med denna klassificerade bild går varje svag kant igenom och kollar om det ansluts till en stark kant. Om det inte gör det turs den kanten bort för att undvika brus eller dåliga konturer.

Denna algoritm är en algoritm som bygger på vanliga bilder och eftersom algoritmen är så gammal är spelkontext inte något som var i åtanke under skapandet av algoritmen. För spelkontext är algoritmen inte best nyttjad i sig för spel eftersom algoritmen använder sig av derivationer som blir svår att räkna i grafikkod. Algoritmen använder även bara in en svartvit bild av scenen och ignorerar faktorer som normaler och depth textures. Normaler är något som DeCarlo, et al. (2003) använder sig av i sin algoritm. Däremot även med den extra information kan Canny algoritmen ofta producera bättre resultat.

I en undersökning av F. Cole, et al. (2008) som kollar på hur nära algoritmerna matchar vad konstnärer i praktiken ritar. Detta ger oss en ovanlig kvantitativ insikt i något som ofta kan bara analyseras kvalitativt. 29 konstnärer får i uppdrag att rita linjer på 3D modeller av ben, mekaniska delar, dukar och syntetiska former. Flera algoritmer jämförs för att kolla hur väl de förutsäger vad konstnärerna ritar. Detta görs genom att jämföra en binär bild av konstnären och hur väl det matchar den binära bilden producerad av algoritmen när de ritar linjer över en halvtransparent bild av objektet. Den konstnärsproducerade bilden går igenom några steg för bättre jämförande och matchning till vad algoritmer producerare. Först blir bilden binäriserad vilket gör att bilden bara har svart och vitt, sen uttunnad till en pixel bredd och nu görs

jämförandet. Jämförande görs också med olika linjedensiteter, för DeCarlo algoritmen betyder detta  $d$  och för Canny algoritmen betyder detta  $T_L$  och  $T_H$ .

De diskuterar att DeCarlo algoritmen kan visa form i en jämn yta och är designad till att förlänga konturer inuti ytan men påpekar att algoritmen producerar inga suggestiva konturer i helt konvexa former eftersom konturer händer endast där radiella kurvan är noll vilket orsakar att ingen vyvinkel kan producera suggestiva konturer på konvexa former.

Canny algoritmen eftersom det baseras på en vanlig bild är den känslig för belysning och skuggning. De tar även upp att algoritmen behöver har noggrant justerade inställningar för att producera bra resultat. Det kan även ha mycket brus och ibland skapa fragmenterade kanter.

Jämfört med varandra producerar Canny algoritmen linjer som närmare matchar vad konstnärer producerar. Författarna tyder på att detta kan vara på grund av att konstnärer använder också själva bilden av objektet som input, de kollar efter ställen med hög kontrast, shading och perceptuella kanter vilket matchar väl med vad Canny algoritmen gör. DeCarlo algoritmen är fortfarande viktigt och har en unik roll i att det skapar linjer oberoende av belysning och skapa geometriska linjer. Det finns ingen entydig algoritm som matchar hur människor perceptuellt tycker linjer ska ritas. De tar upp kombinationer av algoritmer för att skapa mer närliggande linjer och via detta producerar de resultat som närmast ligger till vad människor ritat. DeCarlo algoritmen parar mycket väl ihop med andra algoritmer eftersom det hittar linjer som andra algoritmer missar speciellt hos lina ytor.

Både av dessa historiska artiklar är något som vanligen används som bas i moderna implementationer av konturrendering. F. Cole, et al. (2008) tar också upp en viktig del av modern konturrendering och det är kombination av olika algoritmer för att producera mer korrekta resultat. Avancerad konturrendering bygger sig upp på mer faktorer än vad dessa algoritmer använder. Dessa algoritmer är inte heller designade för spelmotorer och den mer moderna renderingspipeline som finns i de flesta av dagens realtidsapplicationer.

## 4.2 Moderna implementationer

Peng, L. och Shidong, M (2008) presenterar en metod för att stiliserad grafik i stil av kinesiska landskapsmålningar. Denna implementation använder Sobelfilter som bas och använder det för att stilisera grafiken. Sobelfilter är en vanlig metod för att extrahera linjer från bilder, detta är en metod som ofta parar väl ihop med realtidsgrafikpipelines och är snabb. Sobelfilter bygger på två faltningar som går i x och y axeln. Med matriserna av Sobelfilterna kan vi skapa en faltning med en bild där  $I(x, y)$  är en inputbild kan detta göras för att producera kanter. I detta fall är  $T$  det tröskelvärde för att skapa kanter. Författarna av denna text har detta värde satt till 0,07. Författarna tar upp att detta kan användas med annan input än bara själva bilden som kameran ser och därför tar upp att använda detta med djupet av ytan. De multiplicerar  $Z$ -djupet i clip-space steget med en djupskala för att producera en textur som matas in i sobelfiltret. I deras implementation sätter de djupskalan till 0,0045 för att hålla värden så nära mellan 0.0 och 1.0 som möjligt. Detta är faktiskt något som nu efter denna artikel ofta kommer gratis i dagens spelmotorer via Depth Texture som Unity, Unreal Engine och Godot och kan enkelt hämtas eftersom det är en del av renderingspipelinen. Denna metod är en vanlig screen-space metod för att implementera kontureffekter. Exempel på hur Sobelfilter räknas ut för att extrahera linjer kan ses nedanför i figur 3.

$$S_x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$$

$$S_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

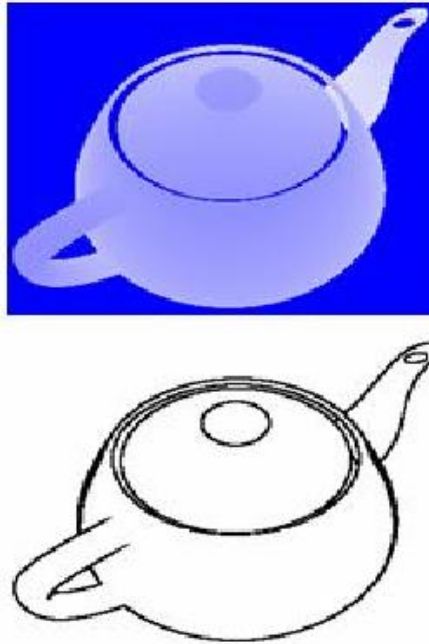
$$I_x(x, y) = I(x, y) * S_x$$

$$I_y(x, y) = I(x, y) * S_y$$

$$I_{mag}(x, y) = \sqrt{I_x^2(x, y) + I_y^2(x, y)}$$

$$(3) \quad Edge(x, y) = \begin{cases} 1 & I_{mag} \geq T \\ 0 & I_{mag} < T \end{cases}$$

Detta skapar binär bild som visar linjerna. I nedanför figur 4 visas vad Sobelfiltret tar emot och under den bilden visas vad Sobelfiltret producerar.



**Figur 4.** Exempel på Sobelfilter från Peng & Shidong (2008)

Vid detta steg gör författarna vissa steg för att få kontureffekten att likna kinesiska landskapsmålningar. För att få linjerna att likna penna och bläck utvidgar de linjerna skapat av Sobelfiltret, författarna använder en noise textur för att utvidga linjerna med slumpmässighet. Noise texturen används som en pensel för att beskriva den texturen utvidgningen händer. För att simulera hur bläck penetrerar papper används Gaussian blur för att mjukna de kanterna skapat av de tidigare steg.

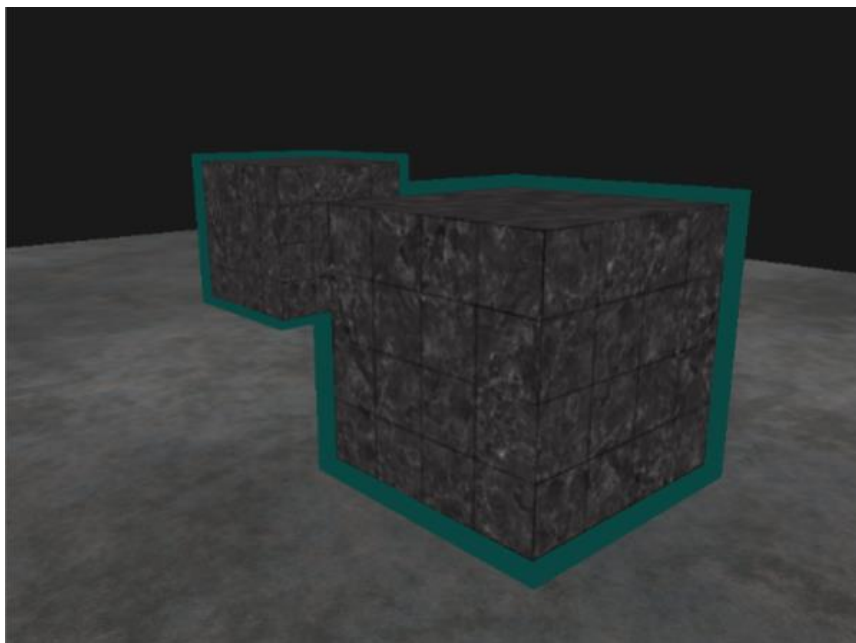
Denna artikel visar en vanlig procedur för skapandet av stiliserade kontur. Det är att använda en redan existerande metod och applicera egna metoder för att uppnå den stilisering som utvecklaren vill ha. Här använder författarna Sobelfiltret för att extrahera kanterna ur kamerans depth texture. Denna metod är populär inom konturrendering eftersom det är ett relativt billigt sätt att extrahera linjer från en bild. Det finns flera filter än Sobelfiter som producerar konturer som Robert's Cross-, Prewitt- och Laplacianfilter. Efter detta modifieras dessa linjer genom en egen skapat metod för att modifiera linjer från linjerna där de expanderar och mjuknar linjer på ett så sätt som liknar kinesiska landskapsmålningar.

Enligt Zheng, S. och Zhong Z (2025) är Sobelfiltret bra på att reducera brus och fungerar väl med en normal map som input. Laplacianfilter är dock bra på att fånga plötsliga skillnader och fungerar väl ihop med ID maps eller depth textures. ID maps är en textur som har en unik färg för varje objekt på skärmen. I deras konturteknik som ska efterlikna Moebius-stilen kombinerar de dessa olika input på olika sätt för att få bakgrunden mindre detaljerad än förgrunden för att få spelaren att fokusera på det som de kan bäst se. De delar upp scener i bakgrund, mellangrund och förgrund. I bakgrunden använder de bara djupet via laplacianfilter, i mellangrunden använder de djupet och ID mappen också via laplacianfilter och i förgrunden använder de djupet, ID:n och normal mappen via laplacian- och Sobelfilter.

De hanterar även karaktärer annorlunda för mer noggrann kontroll över hur konturen ser ut eftersom karaktärer tenderar att ha mer detaljer än miljön. Här de kombinerar backface konturer med ID-map konturer, dessa två konturmetoder ger en effekt som inte har för mycket

eller för lite konturer. Vid detta steg stiliserar de konturer genom att dra de genererade konturer längs normalen i styrkan av en world-space noise textur. Detta följer också samma högnivåprocedur, generera konturer och senare behandla dem för att uppnå önskade stilen.

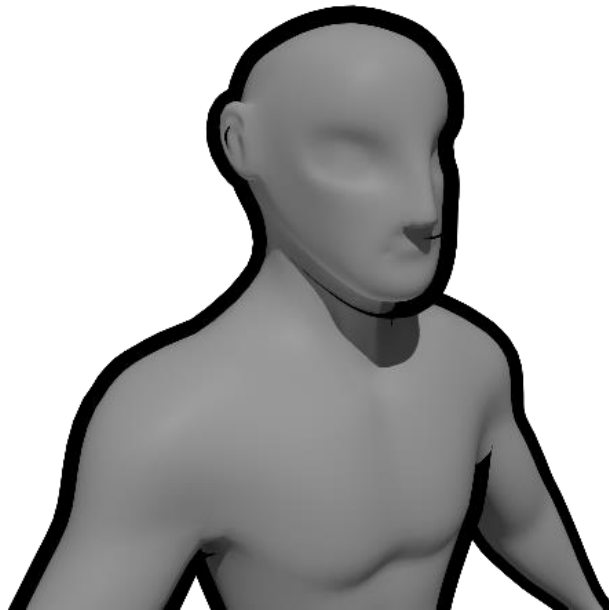
En till screen-space effekt som är vanligt idag är Stencil testing. Detta är en konturmetod som har kommit med industriell bakgrund och har lite akademisk information om. Enligt Vries J. (2020) kan stenciler användas för att producera en kontur runt ett objekt. Stenciler är en buffert som kan användas av fragment shaders för att påverka vad det gör som är i samma storklek som skärmen. För kontureffekter fylls stencilbuffert med ett unikt värde i form av objektet. Detta gör att varje pixel som ritas med objektet har också ett unikt värde på stencilbuffret. Detta tillåter oss att analysera vilka pixlar är nära kanten, om det är det kan en pixel ritas där för att producera en kontur. Detta har flera för- och nackdelar, främst kan detta inte användas för att producera konturer inuti objektet och producerar bara linjer runt objektet och är ofta begränsat i hur många objekt kan ha denna effekt men det är extremt noggrant och anpassningsbar. Detta gör det till bra stöd för större kontursystem men fungerar inte särskilt bra på egen hand. Nedanför i figur 5 visar ett exempel på hur Stencil testing ser ut och vad som händer när två objekt av samma unikt värde överlappar varandra. I vissa fall kan detta vara en önskad effekt och i vissa fall vill detta undvikas.



**Figur 5.** Exempel på Stencil testing av Vries (2020)

Detta täcker mycket av de moderna screen-space metoder för att implementera kontureffekter. För world-space konturmetoder finns det backface konturer som även användes av Zheng, S. och Zhong Z. Enligt Möller, et al. (2018) och är en metod som skapar ett skal av mesh och flytta ut ytorna jäms normalen och vända på sidan av skalets yta. Efter detta sätts alla ytor ihop eftersom vid detta steg finns det massa mellanrum eftersom att flytta ut jäms normalens yta ger luckor mellan två ytor. Detta kan behandlas på olika sätt, författarna tar upp att ha ytorna dela en vertex eller att skapa ny geometri som sedan tars där den möter varandra.

Detta använder sig av backface culling för att ignorera ytor som inte är riktat mot kameran. Detta gör att de ytor som från början pekade bort från kameran pekar mot kameran och eftersom de är expanderade längs normalen kan det ses runt konturer. Detta fungerar väl ihop med vertex shaders vilket gör det speciellt användbart för realtidsgrafik eftersom varje triangel hanteras parallellt och behöver ofta inga extrasteg. De påstår att detta ger en kontureffekt som är effektiv, robust och ger stabil prestanda men har begränsningar i dålig kontroll över kanternas utseende och är knepigt att implementera och fungera väl med halvtransparenta ytor. Nedanför i figur 6 är ett exempel skapat i Blender på hur backface konturer ser ut.



**Figur 6.** Exempel på backface konturer

Detta representerar mycket av de metoder som används i moderna spelsammanhang. Dessa implementationer är inte speciella för den tiden de är skapat i men speciella för de är de metoder som fortfarande används idag i moderna spelapplicationer. Dessa metoder fungerar väl ihop med dagens spelrenderingpipeline vilket gör det optimalt för realtidsapplicationer. Dessa metoder sätter den grunden till nästan alla konturrenderingstekniker i dagen spel.

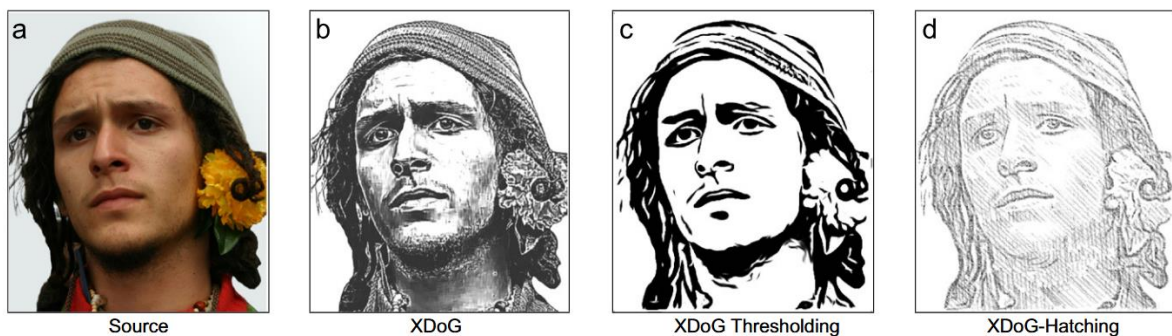
### 4.3 Stiliseringstekniker

För att effektivt använda de generade konturer går de ofta igenom stiliseringssteg som med Peng, L. och Shidong, M (2008) och Zheng, S. och Zhong Z (2025). Det visar sig vara en viktig del för att få konturrendering att se ut exakt som utvecklaren är ute efter. För detta behövs en förståelse av hur dessa produceras för att utnyttja dem på ett effektivt sätt. De sätt som redan har tagits upp är de mest populära sätt att stilisera konturer men det finns även mer unika tekniker.

Zakaria N. och Seidel H. (2004) tar upp en unik teknik för att stilisera konturer. De tar pixlarna från en kontureffekt och återbygger det till linjer genom att kolla vilka pixlar som har en låg vinkelskillnad och representerar en liten skillnad i 3D-rum vilket formuleras till ett polygontåg vilket beskriver linjerna. Pixlar som inte blir anslutna på grund av dess vinkelskillnad eller position blir borttagna. Med dessa polygontåg kan djupare stilisering produceras, linjer kan bli utjämnade och få tjocklek som liknar pennstreck. Detta producerar linjer med hög

anpassningsbarhet med kostnaden av prestanda eftersom denna teknik kräver mycket mer beräkning än andra eftersom det omvandlar pixlar till linjer.

En vidare intressant stiliseringsteknik är difference of Gaussians som turs upp av Möller, et al. (2018). Detta är en teknik där ett screen-space algoritmen appliceras två gånger där bilderna har körts igenom gaussisk blur där styrkan är olika, en av bilderna är subtraherad med denna andra. Detta producerar resultat som ofta producerar tydligare konturer och en mer tyngre linjevikt. Detta kan vara relevant för flera konstnärliga stilar som kan vara passande för bilder som emulerar blyerts eller pasteller. I nedanför figur 7 visas ett exempel på hur en modifierad version av difference of Gaussians ser ut och vad det kan användas till.



**Figur 7.** Exempel på modifierad Difference of Gaussians av Winnemöller, Kyprianidis & Olsen (2012)

För att skapa effektiva konturer i spel används dessa stiliseringstekniker för att producera den önskade still. Detta är långt ifrån alla sätt att stilisera konturer men visar på olika sätt att implementera stilisering som ger en idé av hur detta utförs. Stilisering kan hanteras på många sätt men främst händer i fragment shadern för att ge exakt kontroll över hur kanterna ser ut. Med detta ges en klar bild på hur konturrendering implementeras i moderna spel- och realtidsapplicationer.

## 4.4 Mönsteranalys

För sökningarna inom akademiska källor hittades 83 artiklar där 12 möter kriterierna för analysen. Flera av artiklarna faller utanför datorgrafik eller spelgrafik vilket orsakar en stor mängd exkluderade artiklar. Samma statistik saknas för industriella källor på grund av de enklare sökverktyg och mer irrelevant information.

Författare	År	Ursprung	Renderingstyp	Typ	Realtid
Ivo, R. F., Vidal, C. A., & Cavalcante-Neto	2020	Akademisk	Hybrid	Innovation	Realtid
Liew Suet Fun, I., et al.	2004	Akademisk	Screen-space	Innovation	Realtid
Mukundan, R.	2015	Akademisk	World-space	Innovation	Realtid
Li, P., & Mei, S.	2008	Akademisk	Screen-space	Modifying	Realtid
Chen, D. et al.	2015	Akademisk	World-space	Modifying	Realtid
Liao, J	2023	Akademisk	Screen-space	Översikt	Realtid
Lifeng Xiao, Fengyi Liu, & Yuhui Wang.	2015	Akademisk	World-space	Innovation	Realtid
Nguyen, V., et al.	2024	Akademisk	Screen-space	Innovation	Offline
Bui, M. T., Kim, J., & Lee, Y.	2015	Akademisk	Screen-space	Innovation	Realtid
Dollner, J., Kirsch, F., & Nienhaus, M.	2006	Akademisk	Screen-space	Innovation	Realtid
Bénard, P., Hertzmann, A., & Kass, M.	2014	Akademisk	World-space	Innovation	Realtid
Kang, D., Kim, D., & Yoon, K.	2001	Akademisk	World-space	Översikt	Realtid
Amaye, A.	2021	Industriell	Flera	Översikt	Realtid
Roystan	2025	Industriell	Screen-space	Översikt	Realtid
Zheng, S., Zhong, Z.	2025	Industriell	Hybrid	Innovation	Realtid
PawelM	2014	Industriell	Screen-space	Översikt	Realtid

Det finns betydlig mindre industriella källor i denna tabell eftersom det var betydligt svårare att hitta relevant information från de resurserna som ska ta upp detta. Det finns en stor mängd källor som kan användas för realtidsgrafik. Detta syftar dock inte på att den passar väl i realtidsmotorer. Det finns en stor skillnad på vad som tas upp i akademisk och industriella källor, många av de industriella källorna är handledningar som visar hur tekniker är implementerade medan akademiska artiklar har mycket nya implementationer av konturrendering.

## 5 Sammanfattning och diskussion

### 5.1 Sammanfattning

Icke-fotorealistisk rendering är en central renderingsteknik inom spel, animation, teknisk visualisering och utbildning. Inom spelindustrin används denna typ av rendering inte bara inom stora produktioner men även mindre produktioner.

Inom icke-fotorealistisk rendering är konturrendering ofta en viktig del som försöker emulera den grafiken som ses inom 2D-konst där konturer används för att separera och definiera objekt. I 3D blir detta en mer komplex utmaning eftersom det finns ingen skiss att utgå från. Konturer på modeller behöver alltså bli beräknad utifrån geometrin och kameran vilket skapar en både tekniska och estetiska utmaningar. Hur konturer räknas ut utifrån geometrin och kameran är inte alltid tydlig och konturer i 2D är något som ofta sätts subjektivt i olika ställen och tjocklekar vilket skapar stor variation i hur de ser ut. Detta har gjort att det finns flera metoder för att uppnå konturer som har sina styrkor och svagheter.

Historiskt har handritade illustrationer varit viktiga inom exempelvis medicin, industriell design och akademiska figurer eftersom de ofta kommunicerar tydligare än datorgenererade bilder. Modern forskning har dock förbättrat möjligheterna att återskapa dessa egenskaper digitalt.

För spel används konturer för både estetik och funktion. För spel som Borderlands och Dispatch blir deras användning av kontureffekter en grundläggande del för spelens identitet. Det har även visats att icke-fotorealistisk rendering lyfter spelets konstnärliga uttryck på ett vis som fotorealistisk rendering inte kan. Detta gör att icke-fotorealistisk rendering fyller ett unikt och intressant område för estetik och konstnärligt uttryck.

Konturrendering kan implementeras via flera metoder och är mycket vanligt inom icke-fotorealistisk rendering. Det finns flera metoder som används för att producera konturer alla med sina egna visuella egenskaper och prestandafrågor vilket gör att valet av konturrenderingsmetod är inte alltid ett enkelt svar. Det saknas en tydlig samlad bild av hur olika på hur olika konturmetoder fungerar i moderna sammanhang och hur de används mest effektivt.

Litteraturöversikten som genomförs i denna studie försöker täcka denna informationslucka genom att analysera konturrenderingstekniker utifrån implementation, användning och begränsningar skapas en bild av varför vissa konturtekniker fungerar där andra misslyckas. Teknikerna jämförs utifrån prestanda och visuella egenskaper.

Det finns några metoder som inte passar moderna spelapplicationer men lägger grunden till mer moderna tekniker och visar varför vissa metoder inte fungerar för realtidsrendering. Algoritmen av DeCarlo et al. (2003) är en tidigt geometribaserad metod för att implementera konturer som inte passar bra till realtidsapplicationer eftersom den är beräkningsmässigt komplext och passar inte väl ihop med den moderna realtidsrenderingspipelinen, detta gör att metoden inte är passande för spel. Algoritmen av Canny (1986) är också en klassisk implementation av konturer som händer i screen-space till skillnad från DeCarlo algoritmen.

Problemen med denna metod är dock att sättet objekt är belyst blir en vanlig störningsfaktor och beräkningar som görs är inte speciellt enkelt att implementera i shaders eller liknande vilket gör det svårt för användning i realtidsapplicationer.

Många av de moderna tekniker i dagens spel använder sig av screen-space metoder som Sobelfilter vilket är snabbt och enkelt att integrera med befintliga pipelines eftersom det i grunden är bara två faltningar. Dessa screen-space metoder kan ta emot olika texturer generat från kamera för att generera unika konturer. Dessa metoder kan också kombineras med andra screen-space metoder för att producera konturer som uppnår utvecklarens estetiska krav. Detta skickas vidare till efterbearbetning för att ha exakt den effekt på linjerna som utvecklaren är ute efter via till exempel brus.

Andra vanliga metoder i moderna spel är även stencil testing som är extremt noggranna med konturer men begränsad till yttre kanter och backface konturer vilket är en world-space effekt som är effektiv och stabil men mindre visuellt flexibelt.

Inom industrikällor är det vanligt att se genomgångar som går över implementationsdetaljer hög specificitet. Industriella källor är det dock mer vanligt att nya metoder för konturrendering förekommer.

## 5.2 Diskussion

Konturrendering är en renderingsteknik som förekommer i flera varianter. Alla metoder diskuterat i denna litteraturöversikt fyller ett varierat användningsområde vilket lyfter frågan om vilka faktorer av i utvecklingen av spel påverkar valet eller valen av konturmetoder.

Det finns två större kategorier som metoderna kan placeras i, world-space och screen-space. Dessa två högnivåskategorier har intressanta beteende som kan göra det relevant för vissa implementationer. World-space metoder är unikt i att det ofta automatiskt ändrar storleken baserat på distansen vilket kan vara önskad eller oönskad baserat på hur mycket av skärmen redan är upptaget. Dessa metoder är dock mer knepiga att stilisera på det sätt som är önskad av utvecklaren. Metoderna är dock ofta mycket anpassningsbara när det kommer till inställningar per objekt eftersom effekten ligger på det specifika objektet och är inte en global effekt. Detta gör denna kategori av metoder effektiv för enkla kontureffekter men problematiskt för mer avancerade implementationer. Metoden fyller också en unik nisch inom användargenererat innehåll i plattformar som av 3D modeller eftersom det kräver minimal integrering med den överliggande 3D motorn.

Screen-space metoder täcker de mest populära implementationer av konturtekniker i moderna spel. Detta är för att de är mycket anpassningsbara och kan användas på varierande sätt för att producera kontureffekter som är anpassande för att uppnå estetiska mål och exakt kontroll för att tydliggöra formen av objekt. Denna grupp av metoder blir särskild användbar i situationer där utvecklarna sätter stor fokus i kontureffekter eftersom metoderna i grunden producera inte speciellt bra resultat men blir effektiv i kombination och kreativt användande av texturer genererad från kameran. I helhet ger screen-space metoder mer möjligheter för att uppnå den tydlighet och konstnärligt uttryck som utvecklaren vill ha men är mer avancerat att uppnå

medan world-space metoder producerar ett simpelt men inte flexibelt alternativ.

Detta väcker frågan om när bör tekniker kombineras och när blir detta onödigt komplexitet. Eftersom en eller flera tekniker kan implementeras via ett shaderpass gör att prestandakostnaden är ganska minimal. Detta gör att prestandafrågor för screen-space algoritmer blir endast något som påverkar enheter med lägre prestanda såsom mobiltelefoner och VR system.

Fördelen som kombinationer av konturrendering är att det ofta producerar bättre visuell kvalitet på konturer eftersom det kan täcka mycket mer, till exempel är täcker ID-texturer information som helt saknas från normaler. Detta gör att det är nästan alltid fördelar mer kombinationer av tekniker och inputkällor som tydliggör former vilket är vad G. Winkenbach och D. Salesin (1994) lyfter som en central del av konturrendering.

En viktig faktor är utvecklingstiden och utvecklingsprocessen som kommer från skapandet av effekten och är det största negativet för kombinationer av outlintekniker. För att skapa kontureffekter behöver utvecklaren lista ut vilka effekter gör vad vilket är en reaktionär process vilket komplicerar planeringsstegen som behövs ta för att skapa en enlig visuell stil.

Detta blir en viktig faktor i produktionen av spel med konturrendering eftersom det kräver ofta itererande vilket gör att det blir svårt att skapa en klar definition tidigt vilket är problematiskt eftersom det är något som behöver ha beslut tidigt på grund av dess påverkan på grafiskskapande. Senare utveckling av konturrendering kan skapa risk för inkonsekvent stil. Detta gör att konturrendering kräver en noggrann balans mellan visuell ambition och produktionsprocessen.

Bland mönsteranalysen finns det flera intressanta mönster som förekommer. Även om denna artikel drabbas av få industrireferenser så finns det ett mönster av att flera artiklar är genomgångar över hur tekniker implementeras. Medan akademiska källor kommer med flera fynd, detta är inte en stor överraskning eftersom flera metoder kommer i grunden från akademiska resurser. Fokuset generellt är också inte samma inom dessa områden eftersom inom akademien är nya innovationer högt värderat medan inom spelindustrin är konstnärlig kontroll en stor fokus och de metoder som traditionellt används fungerar väl för dessa ändamål.

Det finns ett överraskande mönster där flera av metoderna skulle kunna fungera i realtid. Det är möjligt att detta är för att konturrendering är del av icke-fotorealistisk rendering vilket i sig fokuserar på simplare representation av objekt vilket ofta betyder att det är betydligt enklare för datorn att räkna ut jämfört med fotorealistisk grafik. Detta kan göra att det sätts större fokus på att konturmetoder ska fungera på realtid. Problematiken med denna kategorisering är dock att det inte representerar metoder som passar bra i moderna renderingspipelines.

## 5.3 Samhälleliga och etiska aspekter

Som diskuterat tidigare behöver utvecklare en bra förståelse över hur dessa teknikers effekt och hur de fungerar. Denna översikt försöker att stödja med denna lucka av information för att hjälpa utvecklare formulera välgrundade beslut. Med bättre information kring konturrendering kan utvecklare mer effektivt producera bättre resultat med konturer vilket möjliggör effekten inom flera områden. Många av dessa konstnärliga effekter och visuell stil blir en fråga om budgetering eftersom effektiviteten av det estetiska valet måste övervägas med tidskostnaden av valet. Om estetiken inte är effektiv men tar lång tid att skapa blir det inte lämpligt för användning i spelproduktioner. Detta gör att en bättre förståelse av konturtekniker gör det billigare att skapa med bättre resultat vilket borde göra texter som hjälper utvecklare att formulera en idé över hur de skapar dessa effekter något som är till att nyttja.

Konturrendering väcker också intressanta kulturella aspekter eftersom det tillåter utvecklare att närma sig kulturella visuella språk i en helt ny miljö. Detta lyfter intressanta kulturella representationer i spel som ofta inte kan ses på grund av skillnader mellan konstmedium. Vi har redan tagit upp Peng, L. och Shidong, M (2008) användning av Sobelfiltret för att producera en effekt som efterliknar kinesiska landskapsmålningar men det finns också flera kulturella viktiga konststilar och konströrelser som använder sig av konst som använder sig av konturer som till exempel Shan Shui, Jugend och Ukiyo-e.

## 5.4 Framtida arbete

Det är svårt att påpeka exakt vart framtida arbete kommer innefatta för konturrendering eftersom det är svårt att veta vilka tekniker som går miste om. Eftersom mycket av utvecklingen av konturrendering är inte speciellt begränsat av prestanda. Mycket av det som begränsar vissa användningar av konturmetoder är arkitekturen av renderingspipeline och inte hur krävande det är beräkningsmässigt. Detta gör att framtida utveckling inom området kommer antagligen innefatta mer stabila och uniforma algoritmer.

Ett intressant område inom spelgrafik som är relativt ny är relevant för konturrendering är geometri shaders vilket bygger ny geometri från vertiser. Kan detta kanske bli använt för att producera mer world-space konturer? Geometry shaders har fördelen att det instansierar den generade geometrin vilket kan göra det mer effektivt i skapandet av konturer.

En sak framtida arbete inom området bör ta upp är mer detaljerade information bakom tankeprocessen när den produceras, speciellt när det kommer till stilisering av konturer. Detta är eftersom det är mer viktigare att förstå processen varför något görs eftersom utvecklare generellt är inte ute efter en specifik lösning utan söker information som tar dom närmare till deras exakta estetiska mål. Spelindustrin i grunden är en industri där konst är oerhört viktigt och därför har antagligen en specifik stil de är ute efter som inte täcks av redan existerande information.

## Referenser

- Amaye, A. (2021). *5 ways to draw an outline*. Tillgänglig på Internet: <https://ameye.dev/notes/rendering-outlines/>
- Bénard, P., Hertzmann, A. and Kass, M. (2014) 'Computing smooth surface contours with accurate topology', *ACM Transactions on Graphics (TOG)*, 33(2), pp. 1–21. Tillgänglig på Internet: <https://doi.org/10.1145/2558307>.
- Bui, M.T., Kim, J. and Lee, Y. (2015) '3D-look shading from contours and hatching strokes', *Computers & Graphics*, 51, pp. 167–176. Tillgänglig på Internet: <https://doi.org/10.1016/j.cag.2015.05.026>.
- Canny, J. (1986). A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol PAMI-8*. 6 uppl., s. 679–698. Tillgänglig på Internet: <https://doi.org/10.1109/TPAMI.1986.4767851>.
- Chen, D. *et al.* (2015) 'Real-Time Artistic Silhouettes Rendering for 3D Models', *2015 8th International Symposium on Computational Intelligence and Design (ISCID), Computational Intelligence and Design (ISCID), 2015 8th International Symposium on*, 1, pp. 494–498. Tillgänglig på Internet: <https://doi.org/10.1109/ISCID.2015.201>.
- Cole, F., Golovinskiy, A., Limpaecher, A., Barros, H. S., Finkelstein, A., Funkhouser, T. & Rusinkiewicz, S. (2008). Where Do People Draw Lines? I *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*. 1st edn. New York, NY, USA, Association for Computing Machinery. Tillgänglig på Internet: <https://doi.org/10.1145/3596711.3596756>.
- DeCarlo, D., Finkelstein, A., Rusinkeiwicz, S. & Santella, A. (2023). Suggestive Contours for Conveying Shape. I *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*. New York, NY, USA, Association for Computing Machinery. Tillgänglig på Internet: <https://doi.org/10.1145/882262.882354>.
- Dollner, J., Kirsch, F., & Nienhaus, M. (2006). Sketchy Illustrations for Presenting the Design of Interactive CSG. *Tenth International Conference on Information Visualisation (IV'06), Information Visualization, 2006. IV 2006. Tenth International Conference On*, 772–777. Tillgänglig på Internet: <https://doi.org/10.1109/IV.2006.97>.
- Ivo, R.F., Vidal, C.A. and Cavalcante-Neto, J.B. (2020) 'Improved silhouette rendering and detection of splat-based models', *Computers & Graphics*, 93, pp. 39–50. Tillgänglig på Internet: <https://doi.org/10.1016/j.cag.2020.09.010>.
- Kang, D., Kim, D., & Yoon, K. (2001). A study on the real-time toon rendering for 3D geometry model. *Proceedings Fifth International Conference on Information Visualisation, Information Visualisation, 2001. Proceedings. Fifth International Conference on, Information Visualisation*, 391–396. Tillgänglig på Internet: <https://doi.org/10.1109/IV.2001.942087>

Lake, A., Marshall, C., Harris, M. & Blackstein, M. (2000). Stylized rendering techniques for scalable real-time 3D animation. I *Proceedings of the 1st International Symposium on Non-Photorealistic Animation and Rendering*. New York, NY, USA, Association for Computing Machinery (NPAR '00), s. 13–20. Tillgänglig på Internet: <https://doi.org/10.1145/340916.340918>.

Li, P. & Mei, S. (2008). A Method Based on Chinese Landscape Painting Style of Non-photorealistic Rendering. I *2008 International Workshop on Education Technology and Training & 2008 International Workshop on Geoscience and Remote Sensing*, s. 743–746. Shanghai, Kina Tillgänglig på Internet: <https://doi.org/10.1109/ETTandGRS.2008.409>.

Liao, J. (2023) 'The Research of Cel-Shading in Non-photorealistic Rendering', *2023 13th International Conference on Information Technology in Medicine and Education (ITME), Information Technology in Medicine and Education (ITME), 2023 13th International Conference on, ITME*, pp. 569–572. Tillgänglig på Internet: <https://doi.org/10.1109/ITME60234.2023.00119>.

Liew Suet Fun, I. *et al.* (2004) 'Non-photorealistic outdoor scene rendering: techniques and application', *Proceedings. International Conference on Computer Graphics, Imaging and Visualization, 2004. CGIV 2004., Computer Graphics, Imaging and Visualization, 2004. CGIV 2004. Proceedings. International Conference on, Computer graphics, imaging and visualization*, pp. 215–220. Tillgänglig på Internet: <https://doi.org/10.1109/CGIV.2004.1323988>.

Lifeng Xiao, Fengyi Liu and Yuhui Wang (2015) 'Research of computer-generated algorithms for traditional Chinese realistic paintings based on 3D model', *2015 Chinese Automation Congress (CAC), Chinese Automation Congress (CAC), 2015*, pp. 767–772. Tillgänglig på Internet: <https://doi.org/10.1109/CAC.2015.7382601>.

Mukundan, R. (2015) 'Multi-level stroke textures for sketch based non-photorealistic rendering', *2015 International Conference and Workshop on Computing and Communication (IEMCON), Computing and Communication (IEMCON), 2015 International Conference and Workshop on*, pp. 1–7. Tillgänglig på Internet: <https://doi.org/10.1109/IEMCON.2015.7344505>.

Möller, T. A., Haines, El., Hoffman, N., Pesce, A., Iwanicki, M. & Hillaire, S. (2018). *Real-Time Rendering, Fourth Edition*. 4 uppl., Taylor & Francis.

Nguyen, V. *et al.* (2024) 'Region-Aware Simplification and Stylization of 3D Line Drawings', *Computer Graphics Forum*, 43(2), pp. 1–15. Tillgänglig på Internet: <https://doi.org/10.1111/cgf.15042>.

PawelM (2014). *Tutorial – Creating outline effect around objects*. Tillgänglig på Internet: <https://www.michalorzelek.com/blog/tutorial-creating-outline-effect-around-objects/>

Roystan (2025). *Outline shader*. Tillgänglig på Internet: <https://roystan.net/articles/outline-shader/>

Steam (u.å.). *Borderlands 2*. <https://store.steampowered.com/app/49520> [2026-04-03]

Steam (u.å.). Return of the Obra Dinn. <https://store.steampowered.com/app/653530> [2026-04-03]

Unity (2017). *Camera's Depth Texture*. <https://docs.unity3d.com/560/Documentation/Manual/SL-CameraDepthTexture.html> [2026-04-02]

Unity (2017). *Using Depth Textures*. <https://docs.unity3d.com/560/Documentation/Manual/SL-DepthTextures.html> [2026-04-02]

Vries, J. (2020) *Learn OpenGL Learn Modern OpenGL Graphics Programming in a Step-by-step Fashion*. Kendall & Welling. Tillgänglig på Internet: [https://learnopengl.com/book/book\\_pdf.pdf](https://learnopengl.com/book/book_pdf.pdf)

Winkenbach, G. & Salesin, D. H. (1994). Computer-generated pen-and-ink illustration. *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA, Association for Computing Machinery (SIGGRAPH '94), s. 91–100.

Winnemöller, H., Kyprianidis, J.E. and Olsen, S.C. (2012) 'XDoG: An eXtended difference-of-Gaussians compendium including advanced image stylization', *Computers & Graphics*, 36(6), pp. 740–753. Tillgänglig på Internet: <https://doi.org/10.1016/j.cag.2012.03.004>.

Zakaria, N. & Seidel, H. P. (2004). Interactive stylized silhouette for point-sampled geometry. I *Proceedings of the 2nd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*. New York, NY, USA: Association for Computing Machinery (GRAPHITE '04), s. 242–249. Tillgänglig på Internet: <https://doi.org/10.1145/988834.988876>.

Zheng, S. & Zhong, Z. (2025). *Generalized Stylized Post-Processing Outline Scheme*. I GDC 2025. San Francisco, CA, USA 17-21 Mars 2025. Tillgänglig på Internet: <https://gdcvault.com/play/1035314/Generalized-Stylized-Post-Processing-Outline>.