Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

GUI Graphical User Interface

ECA Ear-Clipping-Algorithmus

DT Delaunay Triangulation

CDT Constrained Delaunay Triangulation

TM Turingmaschine

CPU Central Processing Unit

FIST fast industrial-strength triangulation framework

SP Steiner Punkte

1 Einleitung

Die größte technische Wende nach der industriellen Revolution war die digitale Revolution gegen Ende des 20. Jahrhunderts.[1] Eingeleitet durch die Entwicklung des Mikrochips und der damit verbundnen Verbreitung des Computers in allen Lebensbereichen, führte sie zu einer dramatischen Veränderung. Nicht nur in der Industire und Produktion findet diese Veränderung statt, welche sich in flexibler Automatisierung äußert, sondern auch in anderen Bereichen. So wird die Entwicklung des Internets durch vernetzte Rechner möglich. Als dann Computer nicht mehr nur in der Forschung und für die automatische Produktion genutzt werden, sondern auch für den täglichen Gebrauch im Büro und daheim geutzt werden können, braucht es grafische Benutzeroberflächen und Betriebssysteme. Doch dort macht die Entwicklung nicht halt. Auch die Unterhaltungsbranche erfährt mit Videospielen eine Revolution, welche ebenso auf Computergrafik angewiesen ist, wie ein einfaches Graphical User Interface (GUI).

Zu Beginn beschränkte sich die Darstellung auf sogenannte ASCII-Art, bei der übliche Zeichen aus dem ASCII-Alphabet benutzt wurden, um komplexe Bilder zu erzeugen. Da dies jedoch nicht genügt, um Flächen und Objekte lückenlos darzustellen, bedurfte es einer Inovation. Obwohl es für Menschen einfach ist, Flächen als ganzes zu betrachten und Polygone in unterschiedlichster Komplexität zeichnerisch darzustellen, ist es für Computer als Ganzes nicht so einfach, diese zu speichern, geschweige denn darzustellen. Flächen und dreidimensionale Objekte kann man, so die Idee, über ihre Eckpunkte (Vertices) und die dazwischen liegenden Kanten (Edges) darstellen. Man erzeugt also ein Polygonnetz, welches den Körper abbildet. Dabei ist die Wahl des Polygons zunächst irrelevant. So könnte man beispielsweise einen Würfel aus Quadraten oder Dreiecken aufbauen, je nach dem wie man die Kanten definiert.[2] In der Praxis sind Polytope und Polygone jedoch meistens unregelmäßig, ergeben sie sich doch zum Beispiel aus Umgebenungsscans mit einem Laser-Scanner oder sind die Oberfläche einer Videospielfigur. Es bietet sich in solchen Fällen nicht an, regelmäßige Polygone, wie Quadrate oder Rechtecke, als Grundlage für das Polygonnetz zu nutzen.

Eine geeignete Methode, um diese komplexen Polygone für Computer effizient darzustellen, ist die Nutzung von Dreiecken als grundlegende Form für die Zerlegung. Dieses Vorgehen bezeichnet man als Triangulation. Diese ist formal die Zerlegung eines topologischen Raumes, hier also eines Polygons in Simplexe. Das Simplex der zweiten Dimension das Dreieck und damit ist die Triangulation ein Verfahren zur Zerlegung eines Poylgons in Dreiecke. Es sei erwähnt, dass es Computern durchaus möglich ist, Flächen und Körper darzustellen, welche nicht aus Dreiecken bestehen. Dies ist jedoch wesentlich speicher- und rechenaufwendiger, als es bei Dreiecken der Fall ist. Für die Darstellung eines Objektes ließe sich alternativ auch eine sogenannte Punktwolke nutzen. Wie der Name bereits

andeutet, wird das Objekt dabei aus einer großen Menge von Einzelpunkten gebildet. Für einen hohen Detailgrad sind dafür allerdings auch sehr viele Punkte nötig, was den Speicheraufwand stark erhöht. Bei der Verwendung von Dreiecken handelt es sich, vorallem bei runden Objekte, eher um eine Approximation der Form. Eine Kugel wäre dann nicht vollständig rund, sondern würde als Polyeder repräsentiert werden. Dadurch spart man jedoch sehr viel Speicherplatz. Man kennt diese optische Vereinfachung aus alten Videospielen, in denen alle Objekte doch recht kantig und spitz aussehen. Man kann auch hier den Detailgrad steigern, indem man die Anzahl der Dreiecke erhöht und ihre Größe senkt. Da Dreiecke Flächen sind, benötigt man von ihnen jedoch eine geringere Anzahl, um ein Objekt darzustellen, als wenn dies mittels einer Punktwolke geschieht.

Um eine Triangulation per Computer durchzuführen, bedarf es eines Algorithmus, welche das Verfahren beschreibt. Von diesen gibt es viele verschiedene, welche unterschiedlichste Herangehensweisen nutzen. Hier seien der ECA, die Delaunay Triangulation (DT) und das Voronoi-Diagramm als Beispiele für solche Algorithmen angeführt. Diese unterscheiden sich erheblich in ihrer Komplexität und Effizienz. Mit einer Laufzeit von $O(n^3)$[5], ist der ECA bei weitem nicht so effizient wie beispielsweise die DT mit $O(n \log n)$. Für den ECA spricht jedoch seine relative Einfachheit im Vergleich zu anderen Algorithmen.

In dieser Arbeit soll jedoch nicht die Laufzeitoptimierung im Vordergrund stehen, sondern etwas anderes. Anschaulichkeit ist ein wichtiger Punkt, wenn es um Didaktik geht. Sie fördert das Verständnis ebenso wie Interaktivität. So hat diese Ausarbeitung zum Ziel eine interaktive Visualisierung für die Triangulation von Polygonen zu schaffen. Dafür ist der ECA aufgrund seiner relativen Einfachheit gut geeignet. Er lässt sich schrittweise durchlaufen und ist somit sehr anschaulich, da in jedem Schritt ein Dreieck der Triangulierung erzeugt wird. Es liegt in der Natur dieses Algorithmus, dass Uneindeutigkeiten auftreten, was die Auswahl des nächsten Dreiecks angeht. Diese führen zum zweiten wichtigen Punkt in dieser Arbeit - der Interaktivität. Der Nutzer der Visualisierungssoftware soll interaktiv entscheiden können, welches das nächste Dreieck ist, welches bearbeitet wird. Er beeinflusst somit direkt das endgülige Resultat. Neben dem dirketen Eingriff des Nutzers, sollen auch Heuristiken genutzt werden können, um diese Auswahl zu treffen. Beispielsweise kann hierfür die Größe des Dreiecks im Bezug auf seinen Flächeninhalt genutzt werden oder auch die Innenwinkel. Es ist angestrebt, dass die Nutzerauswahlen ausgewertet und in eine Heuristik überführt werden. Um dies zu bewerkstelligen, soll die Qualität der Triangulation mittels verschiedener Metriken beurteilt werden. Hierfür kann ein Vergleich zum Voronoi-Diagramm ebenso wie zum Beispiel die Anzahl der sogenannten *Slivers*[3] betrachtet werden. Letztere führen in Anwendungen der Computergrafik oft zu Fehlern, welche vermieden

werden sollten.

Es soll somit nicht nur eine Visualisierung für Triangulationen geschaffen werden, sondern es sollen auch die Auswirkungen einfahcer Heuristiken auf die Qualität dieser Zerlegungen betrachtet werden. Es steht dabei, wie bereits erwähnt, nicht die Laufzeit des Algorithmus im Vordergrund, welche üblicherweise Ziel der Optimierung ist.

2 Theoretische Grundlagen

2.1 Polygone

2.1.1 Definition

Ein geschlossener Streckenzug, also eine Folge von Strecken, welche jeweils einen Endpunkt mit ihrem Vorgänger bzw. Nachfolger gemeinsam haben bilden ein **Polygon**.

Dabei ist es wichtig, dass die Anzahl von Strecken endlich ist. "Das Polygon, [zu Deutsch Vieleck], ist also eine durch eine Folge von Strecken begrenzte ebene Fläche."[10] Das einfachste Beispiel hierfür ist ein Dreicke. Es besitzt die Eckpunkte A, B und C und wir daher vom Streckenzug aus den Strecken $\overline{AB}$, $\overline{BC}$, $\overline{CA}$ begrenzt (s. Abbildung 1). Mit genau diesem Prinzip lassen sich beliebig komplexe Polygone erzeugen und beschreiben. Die Strecken werden auch als **Seiten** und die Endpunkte dieser Strecken als **Ecken** bezeichnet. Es sei angemerkt, dass Kreise, obwohl sie ebenfalls ebene Flächen sind, keine Polygone sind. Das folgt daraus, dass Kreise weder Ecken noch eine Begrenzung aus Strecken besitzen.

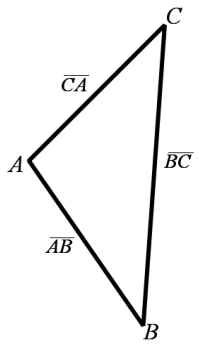


Abbildung 1: Dreick mit Ecken A, B, C als Beispiel für ein Polygon

2.1.2 Klassifikation von Polygonen

Es ist denkbar, dass sich die Seiten des Polygons schneiden oder berühren. Man bezeichnet dieses Polygon als überschlagen.[10] Des weiteren kann man Polygone in regulär und nicht regulär unterteilen. Ein Polygon mit den n Seiten $a, b, c, \ldots$ und den Innenwinkeln $\alpha, \beta, \gamma, \ldots$ heißt regulär, wenn

$$a = b = c = \ldots \text{ und } \alpha = \beta = \gamma = \cdots$$

gilt. In einem regelmäßigen Polygon sind demnach alle Seiten zueinander kongruent und alle Winkel gleich groß. [11]

Eine weitere Unterteilungsmöglichkeit lautet wie folgt. Ein Polygon heißt **konvex**, wenn für alle Innenwinkel α_i ($i \in \mathbb{N}$) gilt: $\alpha_i < 180°$ Anderenfalls heißt es **konkav**. [12]

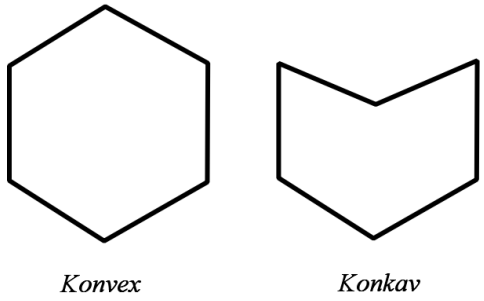


Abbildung 2: Zwei Secksecke: links konvex, rechts konkav

2.1.3 Diagonalen

Außer des Streckenzuges, welcher die äußere Grenze des Polygons bildet, kann man im Polygon selbst auch weitere Strecken definieren, welche dann als **Diagonalen** bezeichnet werden. Mittels dieser Diagonalen ist es möglich, jedes Polygon in Dreiecke zu zerlegen. Das wird in den nächsten Kapiteln noch näher erläutert, da dieser Sachverhalt die Grundlage für sämtliche Zerlegungsalgorithmen darstellt.

2.1.4 Ear und Ear Tips

Für den in Kapitel 3.3 beschreibenen ECA ist es der Begriff des **Ear** (Ohr) relevant. Ein Dreieck, welches aus drei aufeinanderfolgenden Ecken $v_{i_0}, v_{i_1}, v_{i_2}$ des Polygons gebildet wird, ohne dass andere Ecken innerhalb dieses Dreiecks liegen oder dass der äußere Streckenzug des Polygons durch die Seiten des Dreiecks geschnitten wird, nennt man Ear. Dabei soll die Strecke $\{v_{i_0}, v_{i_2}\}$ eine Diagonale des Polygons. Die Ecke v_{i_1} heißt dann **Ear Tip**.[20]

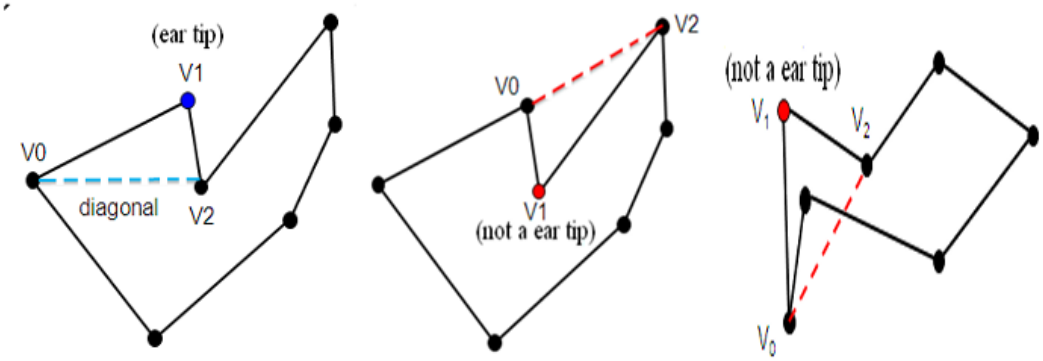


Abbildung 3: Links ist $\triangle v_{i_0}, v_{i_1}, v_{i_2}$ Ear und v_{i_1} Ear Tip. In der Mitte und rechts ist $\triangle v_{i_0}, v_{i_1}, v_{i_2}$ kein Ear, da $\{v_{i_0}, v_{i_2}\}$ keine Diagonale ist. [9]

2.1.5 Sätze über Polygone

Für Polygone gibt es einige Erkenntnisse, welche für die allgemeine Strukturanalyse von eben diesen oder auch für die Triangulation mittels ECA von Bedeutung sind.

Satz 1 (Jordan'scher Kurvensatz):
In der euklidischen Ebene $\mathbb{R}^2$ zerlegt jede geschlossene Jordan-Kurve $C \subset \mathbb{R}^2$ deren Komplement $\mathbb{R}^2 \setminus C$ in zwei disjunkte Gebiete, deren gemeinsamer Rand die Jordankurve C ist und deren Vereinigung zusammen mit die ganze Ebene $\mathbb{R}^2$ ausmacht.
Genau eines der beiden Gebiete, das sogenannte **Innengebiet**, ist eine beschränkte Teilmenge von $\mathbb{R}^2$.
Das andere dieser beiden Gebiete ist das sogenannte **Außengebiet** und unbeschränkt. [17]

Satz 2 (Dreieckszerlegung:)
Jedes Polygon P mit n Ecken kann mittels hinzunahme von null oder mehr Diagonalen vollständig in Dreiecke zerlegt werden. [9]

Satz 3 (Anzahl der Diagonalen:)
Jede Triangulation eines Polygons P mit n Ecken besteht nutzt $(n-3)$ Diagonalen und besteht aus $(n-2)$ Dreiecken. [9]

Satz 4 (Two Ears Theorem:)
Jedes Polygon P mit $n \geq 4$ Ecken besitzt mindestens zwei nicht überlappende Ears. [19]

2.1.6 Polytope

Zuletzt sei an dieser Stelle angemerkt, dass ein Polygon die zweidimensionale Ausprägung des topologischen Begriffs des **Polytops** ist. Betrachtet man die räumlichen Dimensionen null bis vier in aufsteigender Reihenfolge so sind ein Punkt, eine Strecke, ein Quadrat, ein Würfel und ein Tesserakt. Dies ist in der nachstehenden Abbildung zu sehen.

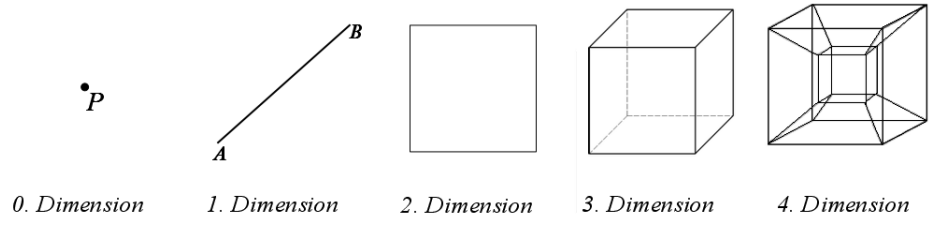


Abbildung 4: Beispiele für Polytope der Dimensionen 0 bis 4

2.2 Simplexe und Triangulation

Wie bereits angesprochen kann man durch hinzufügen von Diagonalen in einem Polygon, dieses in Dreiecke oder allgemeiner in Unterpolygone zerlegen. Diese Eigenschaft macht sich die **Triangulation** zu nutze. Allgemein beschreibt der Begriff Triangulation die Zerlegung eines topologischen Raumes in **Simplexe**.[13] Der topologische Raum ist in diesem Fall das Polygon, welches durch einen Streckenzug gebildet wird.

Als Simplex bezeichnet man das einfachste Polygon einer Dimension.[14] Für die nullte Dimension ist das trivialer Weise der Punkt. Da keine räumliche Ausdehnung möglich ist, ist der begrenzende Streckenzug hier nur der Punkt selbst. In der ersten Dimension, in welcher Objekte eine Länge aber keine Breite besitzen, ist der Streckenzug eine einzelne Strecke. Diese ist somit auch das Simplex dieser Dimension. Für die zweite Dimension ist nun das Dreieck das Simplex. Es ist die Fläche, welche aus den wenigsten Punkten, verbunden duch Strecken, erzeugt werden kann und daher das einfachste Polygon dieser Dimension.

Wie bereits beschrieben, kann jedes komplexere Polygon so durch Diagonalen zerlegt werden, dass es volständig von Dreiecken repräsentiert wird. Das ist besonders günstig für eine Bearbeitung durch Computer, da ein Dreieck immer eindeutig durch seine drei Eckpunkte beschrieben wird. Diese Form ist daher speichereffizienter als beispielsweise ein allgemeines Viereck, für welches nicht nur die Ecken, sondern auch die Kanten gespeichert werden müssten, da mehr als eine Möglichkeit existiert diese vier Punkte zu einem Streckenzug zu verbinden.

Da in der Praxis nicht nur zweidimensionale sondern auch dreidimensionale Objekte eine Rolle spielen, stellt sich die folgende Frage. Kann man diese 3D-Objekte nicht auch in ebenfalls dreidimensionale Simplexe zerlegen?

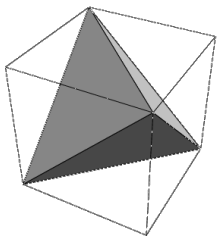


Abbildung 5: Zerlegung eines Würfels in vier Tetraeder [18]

Die Antwort ist ja, jedoch ist das nicht sonderlich nützlich. Natürlich existiert in der dritten Dimension auch ein Simplex. Dieses ist der Tetraeder. Man kann auch jedes Polytop dieser Dimension in Tetraeder zerlegen, nur benötigt man diese Zerlegung nicht. Wöllte man das Innere eines Objektes durch die Zerlegung ebenfalls erhalten, beispielsweise einen Vollwürfel aus Holz in der Realität zersägen, dann wäre eine Tetraederzerlegung notwendig. Ein Beispiel für diese Art der Zerlegung ist in Abbildung 5 zu sehen. Da man in der digitalen Welt keine echten ausgefüllten Volumina betrachtet sondern nur die Oberfläche darstellt, ist dieses verfahren für die Computergrafik uninteressanrt. Man kann die räumlich orientierte Oberfläche eines Würfels topologisch isomorph zu einem Würfelgitter aus quadratischen Flächen beschrieben. Diese Gesamtfläche lässt sich dann wiederum in Dreiecke zerlegen. Somit lässt sich auch die Oberfläche dreidi-

mensionaler Objekte durch eine Triangulierung beschreiben. Das ist besonders gut geeigent für Computer, da man nur einen einzigen Algorithmus zur Bearbeitung von Flächen und Körpern benötigt, wenn man diese in Dreiecke zerlegen möchte.

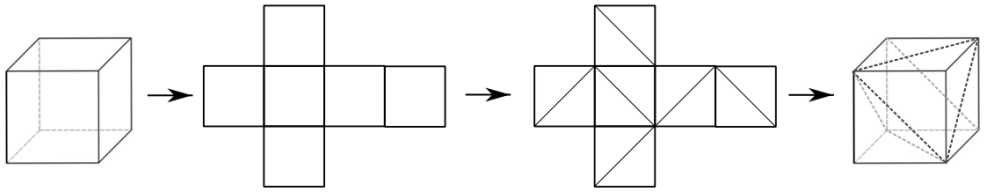


Abbildung 6: Zerlegung der Oberfläche eines Würfels in Dreiecke mittels Würfelgitter

2.3 Slivers

Bei Triangulationsalgorithmen wie dem ECA liegt der Fokus zunächst nicht in der Qualität der Zerlegung. Natürlich gibt es verbesserte Versionen wie beispielsweise in Kapitel 3.1 beschreiben wird. In jedem Fall sind sogenannte **Slivers** jedoch ein negativer Einflussfkator, wenn es um qualitative Gesichtspunkte geht. Dreiecke haben in der Computergrfaik die Eigenschaft, dass wenn man eine Scanline durch das Dreieck legt, dann schneidet diese ein Dreieck zweimal die Kanten des Dreieck. Diese zwei Schnittpunkte, werden von zwei unterschiedlichen Pixeln auf dem Bildschirm repräsentiert. Somit lässt sich definieren, wo eine Fläche beginnt und wo sie endet und dies auf dem Bildschirm darstellen. Bei Slivers ist das nicht der Fall. Ihre Innenfläche ist so schmal, dass die beiden Schnittpunkte der Scanline mit den Kanten des Dreiecks auf den selben Pixel fallen. [4] Das führt in letzter Instanz zu Grafikfehlern.

Der Begriff Sliver beschränkt sich nicht nur auf Dreiecke. Auch andere Simplexe wie Tetraeder können Simplexe sein. Dass sind diese so flach, dass auch hier Darstellungsfehler entstehen. Zusätzlich dazu gibt es noch den verwandten Begriff der **Neadle**, der in Bezug auf Tetraeder, einen sehr schmalen aber auch sehr spitzen solchen bezeichnet.[3]

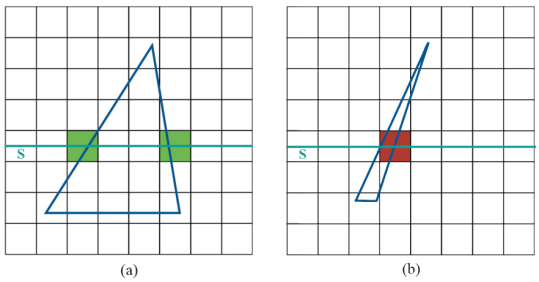


Abbildung 7: (a) Scanline mit zwei seperaten Schnittpunktpixeln (b) Sliver mit nur einem Pixel für beide Schnittpunkte

2.4 Der Traditionelle Ear-Clipping-Algorithmus

Der, in dieser Arbeit, zentrale Punkt, ist der ECA. Dieser wurde von Meister in seiner Abhandlung *Polygons have ears* [20] in seiner ursprünglichen Form beschrieben. Der Algorithmus bestimmt Dreiecke, welche die Eigenschaft eines Ears erfüllen, fügt die dafür nötige Diagonale in eine Liste ein und löscht den Punkt, welcher ein Ear Tip ist aus der Liste aller noch nicht bearbeiten Punkte. Dies wird solange wiederholt, bis das Restpolygon nurnoch aus drei Punkten besteht. Zuletzt wird die Lister der Diagonalen ausgegeben, da diese die Triangulation des Polygons erzeugt. Der Algoritmus ist im Folgenden in Pseudocode dargestellt.

Algorithmus 1: Traditionelles Ear-Clipping [8]

Eingabe: Polygon P mit n Ecken in einer Liste L
Ausgabe: Liste D mit $n-3$ Diagonalen, die eine Triangulierung bilden.
Schritt 1: Sei $D := \emptyset$ Liste der Diagonalen.
Schritt 2: **while** $|L| > 3$ **do**
(a) Finde ein Ear v_{i-1}, v_i, v_{i+1}
(b) $D := D \cup \{v_{i-1}v_{i+1}\}$
(c) $L := L \setminus v_i$
endwhile
Schritt 3: Ausgabe von D als triangulierende Diagonalen.

Dieser Algoritmus hat einen Zeitaufwand von $O(n^3)$ mit einem Aufwand von $O(n^2)$ für das Ermitteln des Ear-Status eines Dreiecks. In dieser Formulierung wird nicht auf die Klassifikation eines Ears im speziellen eingegangen. Hierfür beginnt man klassisch beim ersten ersten Punkt in L. Man überprüft ob dieser Punkt v_i konvex ist. Ist das der Fall, dann muss die Strecke $\{v_{i-1}v_{i+1}\}$ die Eigenschaft haben, eine Diagonale von P zu sein. Wenn das ebenfalls zutrifft, dann ist das Dreicke v_{i-1}, v_i, v_{i+1} ein Ear. Man kann die Klassifikation so darstellen:

Algorithmus 2: Ear Klassifikation

Eingabe: Ecken $v_i, v_{i-1}, v_{i+1} \in L$
Ausgabe: $\triangle v_{i-1}, v_i, v_{i+1}$ ist Ear oder nicht.
Schritt 1: **if** v_i konvex
if $\{v_{i-1}v_{i+1}\}$ ist Diagonale von P
Ausgabe $\triangle v_{i-1}, v_i, v_{i+1}$ ist Ear.
else Ausgabe $\triangle v_{i-1}, v_i, v_{i+1}$ ist kein Ear.
endif
else Ausgabe $\triangle v_{i-1}, v_i, v_{i+1}$ ist kein Ear.
endif

Diese Klassifikation könnte man auch zuerst über alle Punkte v_i in L laufen lassen. Man spricht dann von der Klassifikationsphase. Danach kann man in einer zweiten Phase, der Cutting-Phase, Dreicke auswählen, welche die Ear-Eigenschaft erfüllen und sich nicht überschneiden, und diese dann Abschneiden. Mit diesen beiden Phasen im Wechsel kann man ebenfalls eine Triangulation erreichen. O'Rourke beschreibt in seinem Buch einen Ansatz, der einige Zeitersparnis bei diesem Algorithmus bewirkt.[22] Anstatt nach dem Abtrennen eines Ear Tip Punkts den Status jedes Eckpunktes erneut zu überprüfen, muss man nur den Status von v_{i-1} und v_{i+1} erneut betrachten. Nur diese beiden Punkte sind nämlich vom Abtrennen von v_i beeinflusst. Somit benötigt man insgesamt nurnoch eine Zeit von $O(n^2)$.[9] In jedem Cutting-Schritt kann man dann zusätzlich entscheiden, welche Dreieck als nächstes ausgewählt werden soll. So könnte an nur die Dreiecke auswählen, welche einer bestimmten Heuristik entsprechen. Beispielsweise könnten so nur Dreiecke gewählt werden, bei denen der kleinste Innenwinkel das Maximum aller aktuell verfügbaren Innenwinkel ist. Auf diese Weise ist es denkbar, dass man Sliver vermeiden könnte. Andere Ansätze sind exemplarisch im folgenden Abschnitt aufgeführt.

3 Verwandte Arbeiten

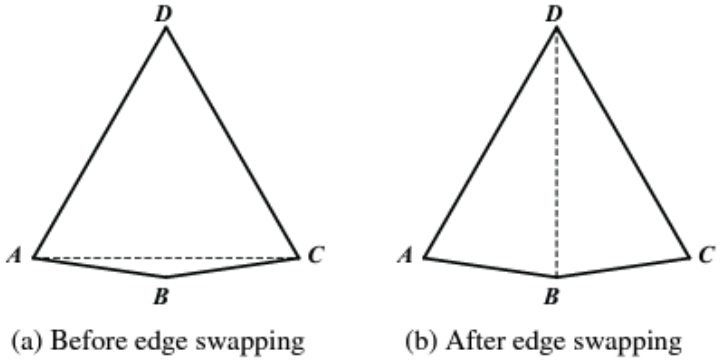
3.1 Verbesserungen des Ear-Clipping-Algorithmus

Wenn man einen Algorithmus mathematisch betrachtet, dann ist die Zeit, welcher er bis zur Terminierung benötigt, zumeist der Gegenstand der Betrachtung. Mittels der Komplexitätstheorie lässt sich diese, vom Typ des Computers, auf welchem der Algorithmus läuft, unabhängig, beschreiben. Das Referenzmodell ist dabei zumeist die Turingmaschine (TM).[21] Ziel ist es, dass ein Algorithmus auf einer TM in Polynomialzeit abläuft. Diese Zeit hängt zumeist von der Eingabegröße ab. Für den ECA ist die entscheidende Größe die Anzahl der Ecken n des Polygons P. Betrachtet man den ECA auf einer TM, so hat er eine Komplexität von $O(n^3)$. Zwar ist dieser Term ein Polynom in n, jedoch ist das kein Grund für die Wissenschaft, hier mit der Optimierung aufzuhören. Wie O'Rourke in seiner Arbeit zeigt, kann man den ECA durch kleine Änderungen so modifizieren, dass dieser eine Komplexität von $O(n^2)$ aufweist.[22] Diese Erkenntnis nutzen Mei, Tipper und Xu, um den Algorithmus auf andere Art zu verbessern.[15]

Für einen Algorithmus wie den ECA ist nicht nur seine Laufzeit entscheidend. Während andere Algorithmen beispielsweise Entscheidungsprobleme lösen, bei denen es nur um die Frage nahc der Existenz der Lösung geht, ist bei einer Triangulation bereits bekannt, dass es unterschiedliche Lösungen gibt. Daher ist die Frage nicht, ob eine Lösung existiert, sondern ob eine optimale solche gefunden werden kann. Optimal ist dabei ein relativer Begriff, der stark von den Rahmenbedingungen abhängt. Für den ECA ist der Speicherbedarf ebenfalls entscheidend. Dieser ist bei Mei, Tipper und Xu durch $O(n)$ begenzt. Das Ziel ihrer Arbeit war es, qualitytiv hochwertige Triangulationen für komplexe Polygone zu erzeugen. Der Qualitätsparameter war dabei der kleinste Innenwinkel der erzeugten Dreiecke in der Zerlegung. Genauer ging es darum, die sogenannten Slivers zu vermeiden (s. Kapitel 2.3).

Ihr Ansatz war es, den verbesserten Algorithmus von O'Rourke so zu modifizieren, dass er über die Option des **Edge Swappings** verfügt. Wird ein Ear erkannt, dann wird für jeden seiner Innenwinkel überprüft, ob dieser kleiner ist als ein zuvor festgelegter Grenzwert. Ist das der Fall, dann muss bei diesem Dreieck Edge Swapping durchgeführt werden. Dafür werden der größte Innenwinkel des Dreickes und die, ihm gegenüberliedende, längste Seite bestimmt. Daraufhin wird überprüft, ob es ein Nachbardreieck gibt, welches sich mit dem Ursprünglichen eben diese längste Seite teilt. Gibt es einen solchen Nachbarn, dann wird in dem von den beiden Dreiecken gebildeten Viereck eine Diagonale zwischen den beiden Ecken gezogen, welche jeweils der zuvor bestimmten längsten Seite gegenüber lagen. Auf diese Art entstehen wieder zwei Dreiecke, von denen nun die Innenwinkel auf ihre Größe überprüft werden. Ist jeweil der kleinste Winkel größer als der

Grenzwert, dann ist die Qualität der Dreicke nun besser als die des ursprünglich Ausgewählten. Wenn dem nicht so ist, bleibt alles unverändert und das Ear wird wie es war gewählt und abgeschnitten.



(a) Before edge swapping (b) After edge swapping

Abbildung 8: Edge Swapping [15]

Auf diese Art kann die Qualität der Dreiecke in der Triangulation stark erhöht werden. Sie zeigen an Beispielen, dass sich die Innenwinkelgröße durchschniitlich verdoppelt. Teilweise können Dreiecke mit minimalem Innenwinkel von $< 15°$ ganz eliminiert werden. Das hängt jedoch vom eingegebenen Poylgon ab und hat keine Allgemeingültigkeit.

3.2 Parallelisierung des Ear-Clipping-Algorithmus

Anstatt den Algorithmus selbst in seiner Laufzeit zu verbessern, ist es ein Gedanke, die Abarbeitung aufzuteilen. Vorallem mit der technischen Entwicklung mehrerer Prozessorkerne in einer Central Processing Unit (CPU), ist verteiltes Rechnen ein gängiges Konzept. Hierzu haben Eder, Held und Palfrader eine Arbeit verfasst, die sich mit der Umsetzung des ECA unter dem Gesichtspunkt der coarse-grain parallelization, zu Deutsch grobkörnigen Parallelisierung, befasst.[7] Dieses Prinzip beschreibt die Aufteilung eines Programms in längere Unteraufgaben. Das ist ein für Multicore Computer sehr geeignetes Konzept. Andere Arbeiten befassten sich auch mit der Umsetzung der Arbeitsteilung aber dort speziell mit dem Konzept der fine-grain parallelization im Bezug auf die DT. Die fine-grain parallelization, also die feinkörnige Parallelisierung, beschreibt die Aufteilung eines Programms in eine Vielzahl kleinerer Aufgaben. Hier ist beispielsweise M. Goodrich[23] zu nennen.

Eder, Held und Palfrader haben ihre Arbeit auf FIST aufgebaut. Dieses Framework ist ein in C++ verfasster Code für Polygontriangulation basierend auf dem ECA.[7] Sie beschränkten sich dabei mit der Parallelisierung auf den Bereich des Algorithmus, welcher sich mit der Klassifikation und dem Clipping der Ears befasst. Dieser Teil mach etwa 80% des Rechenaufwandes aus. Um eine Aufteilung in k Threads zu erreichen, welche dann auf den k Kernen der CPU

abgearbeitet werden sollen, nutzen Sie drei verschiedene Ansätze und vergleichen diese miteinander und mit der nicht parallel laufenden Form des Algorithmus in FIST.

Ihr erster Ansatz beruht auf dem *devide-and-conquer-Prinzip*. Anstatt das Polygon P allerding durch Diagonalen in etwa gleich große Unterpolygone P_k zu unterteilen, nutzen Sie $k-1$ viele senkrechte Geraden dafür. Dies ist weit weniger aufwendig in der Berechnung, da das Finden von geeigneten Diagonalen relativ rechenintensiv ist. Sie berufen sich dabei auf einen Algorithmus von Sutherland und Hodgman [25]. Bei dieser Form der Unterteilung enstehen sogenannte Steiner Punkte (SP), welche die Schnittpunkte der senkrechten Geraden mit den Strecken der äußeren Begrenzung darstellen. Dafür benötigt man eine Zeit von $O(n)$ pro Gerade l und fügt im schlimmsten Fall $O(n)$ SP ein. Diese werden als neue Eckpunkte in den Unterpolygonen eingefügt und damit vom ECA auch als Eckpunkte der Dreicke in der Zerlegung benutzt. Das führt dazu, das Dreiecke in der Gesamtzerlegung von P entstehen, welche unzulässige Eckpunkte besitzen, da diese im Ursprünglichen Polygon nicht existieren. Dafür muss eine Bereinigung der Zerlegung durchgeführt werden, nachdem alle k Threads ihre Triangulation der P_k Unterpolygone geliefert haben. Durch den Schnitt des Poylgons mit einer senkrechten Geraden entstehen zwei SP s_a und s_b. Um diese wieder zu löschen, werden alle Dreiecke, welche zu einem dieser beiden Punkte inzident sind, aus der Triangulation gelöscht. Auf diese Weise erzeugt man ein Loch H in der Zerlegung, welches wieder ein Polygon ist. Diese kann man nun durch erneute Triangulation mit validen Dreiecken füllen.

Einen *partition-and-cut-Ansatz* zu verwenden, war die zweite Variante, um die Triangulation auf die k Threads aufzuteilen. Dabei wird nicht wie bei *divide-and-conquer* das gesamte Polygon P, sondern nur sein begrenzender Streckenzug unterteilt. Hierfür werden **Landmarks**, also Wegpunkte, eingeführt. Dies geschieht anhand der Indizierung der n Ecken. Die Eckpunkte mit den Indizes $\left\{0, \frac{n}{k}, \frac{2n}{k}, \ldots, \frac{(k-1)n}{k}\right\}$ werden die Landmarks. Der Streckenzug zwischen je zwei dieser Markierungen wird jeweils einem Thread zugewiesen. Die Wegpunkte gehören dabei jeweils zu zwei benachtbarten Teilstreckenzügen gleichzeitig. Jeder Thread durchläuft dann eine Klassifikations- und eine Clippingphase, bei denen darauf geachtet wird, dass die Landmarks nicht gelöscht werden. Ist das geschehen und alle Threads beendet, dann bleibt ein Teil des Polygons noch unbearbeitet. Dieser Teil wird bei Eder, Held und Palfrader nicht nocheinaml in Abschnitte für verschiedene Threads unterteilt sondern wird dann vom sequentiellen ECA in FIST bearbeitet. Zwischen den Threads wird keine Synchronisation benötigt, da sowohl Klassifikation als auch Clipping völlig undabhängig von anderen Threads ablaufen und nur in ihrem jeweiligen Abschnitt Dreiecke erzeugt werden. Dabei sei angemerkt, dass das Überprpüfen der Ear-Eigenschaft nur Lesezugriff auf die

Globale Liste aller Eckpunkte des Polygons benötigt. Der Vorgang der Aufteilung in Threads und deren bearbeitung ist in der nachstehenden Abbildung zu sehen.

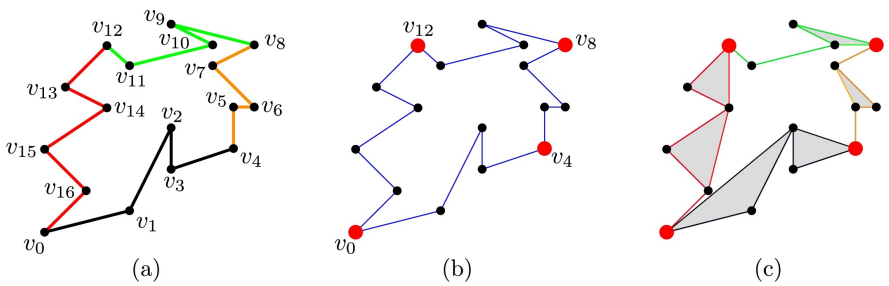


Abbildung 9: (a) Einfaches Polygon P unterteilt in vier Streckenzüge (b) Landmarks vervorgehoben (c) Triangulierung durch Threads

Der dritte Ansatz, betreffend dem ECA in FIST ist der sogenannte *mark-and-cut-Ansatz*, welcher Ähnlichkeiten zum vorher erwähnten *partition-and-cut-Ansatz* aufweist. Auch in diesem Fall werden einige Eckpunkte von P als Markierungen genutzt. In der Marierungsphase, durchläuft ein Thread den Streckenzug von P und speichert jeden zweiten konvexen Eckpunkt in einer Liste. Hat dieser Thread die Hälfte aller Punkte überprüft, werden die Cut-Threads gestartet, welche nur die Cutting-Phase durchlaufen. Bildet ein Punkt in der Liste mit seiner gegenüberliegenden Seite ein Dreicke, dann wird dieses sofort als valide gespeichert und abgeschnitten. Jeder dieser Punkte darf nur einmal bearbeitet werden, damit es nicht zu Asynchronität und Redundanz kommt. Wenn die Cut-Threads alle ihre Arbeit getan haben, werden sie neu gestartet und bearbeiten dann alle Punkte, die Seit ihrem letzten Start zur Liste hinzugefügt worden sind. Während dessen durchläuft der Mark-Thread das Polygon erneut und fürgt neue konvexe Punkte zu Liste hinzu und so weiter, bis nurnoch weniger Dreiecke erkannt werden, als vorher mit einem Grenzwert festgelegt. Dieser lag bei Eder, Held und Palfrader bei 20 Dreiecken. Der Rest von P, welcher noch nicht bearbeitet wurde, wird dann wie im *partition-and-cut-Ansatz* von einem sequentiellen Aufruf von FIST bearbeitet.

Der finale Vergleich aller drei Ansätze hinsichtlich ihrer Qualität zeigt, dass sie

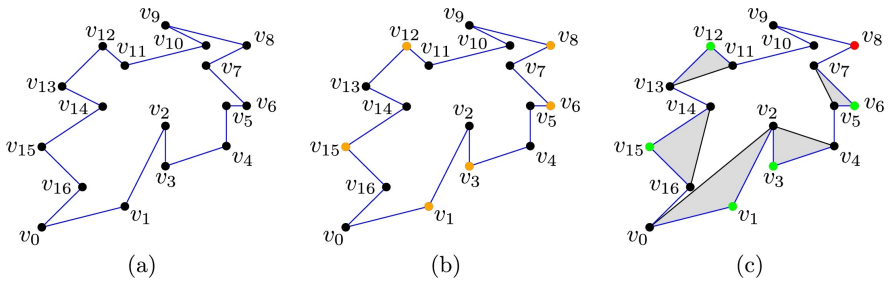


Abbildung 10: (a) Einfaches Polygon P (b) Erste Markierungsphase, ausgewählte Ecken in orange (c) Erste Cutting-Phase

in etwa gleich gut sind. Als Vergleich wurde von ihnen noch eine Variante der DT, die sogenannte Constrained Delaunay Triangulation (CDT) durchgeführt, auf die an dieser Stelle allerdings nicht genauer eingegangen werden soll. Die folgende Tabelle zeigt das Ergebnis.

	CDT	FIST's top	D&C	P&C	M&C
(1) Durchschn. Abw. 60°	30.79°	31.53°	35.29°	34.97°	38.38°
(2) Durchschn. min. Winkel	24.60°	23.40°	20.07°	21.32°	21.07°

Tabelle 1: Vergleich der FIST Trianulation inklusive CDT und FIST's top Heuristik. Aufgeführt sind folgende Parallele Versionen von FIST: Divide-and-conquer D&C, Partition-and-cut P&C und Mark-and-Cut M&C. (1) Durchschnittliche Abweichung aller Innenwinkel von 60° über alle Triangulationen (je kleiner, desdo besser) (2) Durchschnittliche Größe des kleinsten Innenwinkels aller Dreicke über alle Trinagulationen (je größer, desdo besser) [7]

3.3 Alternative Verfahren und Ansätze

3.3.1 Delauny Triangulation

3.3.2 Algorithmus basierend auf Sichtbarkeit

Wie schon bei der DT ist hier der Begriff der Sichtbarkeit von Eckpunkten entscheidend. Anders als zuvor werden hier jedoch nicht sofort Dreicke ermittelt, welche die Bedingung des leeren Umkreises erfüllen. In diesem Algorithmus, beschrieben von Ran Liu, wird das Polygon P in zwei Unterpolygone P_1 und P_2 unterteilt, welche dann solange rekursiv weiter unterteilt werden, bis die entstandenen Unterpolygone P_i Dreiecke sind.[9] Hierfür muss in einem ersten Schritt die Sichtbarkeit jedes Punkte gegenüber einem Referenzpunkt v_i überprüft werden. Dazu betrachtet man zunächst die Nachbarn v_{i-1} und $v_i + 1$ von v_i. Diese begrenzen das sogenannte **Sichtfeld** von v_i, welches mit α bezeichnet wird und dem Winkel in v_i entspricht. Für spätere Betrachtungen zählen v_{i-1} und v_{i+1} als nicht sichtbar im BEzug auf v_i. Entlang der Kanten von P wird nun ausgehend von v_{i+1} entgegen dem Uhrzeigersinn überprüft, ob ein Punkt v_j im Sichtfeld von v_i liegt. Ist das der Fall, dann gilt der Punkt v_j als sichtbar, wenn die Strecke $v_i v_j$ eine Diagonale von P ist. Zusätzlich begrenzt dieser Punkt nun das Sichtfeld und es muss verkleinert werden. Das neue Sichtfeld α berchnet sich also durch $\alpha = v_{i-1}, v_i, v_j$. Der Vorgang wird fortgesetzt, bis alle Punkte überprüft sind. Ist diese Überprüfung für alle v_i von P abgeschlossen, wird die Anzahl der sichtaberen Punkte gegenüber dem jeweiligen Referenzpunkt bestimmt. Diese dient als Vergleichskriterum für die Punkte untereinander. Als Beispiel ist in der nachfolgenden Abbildung einmal die Ermittlung von *alpha* und die Überprüfung mehrerer Punkte bezogen auf den Punkt v_0 dargestellt.

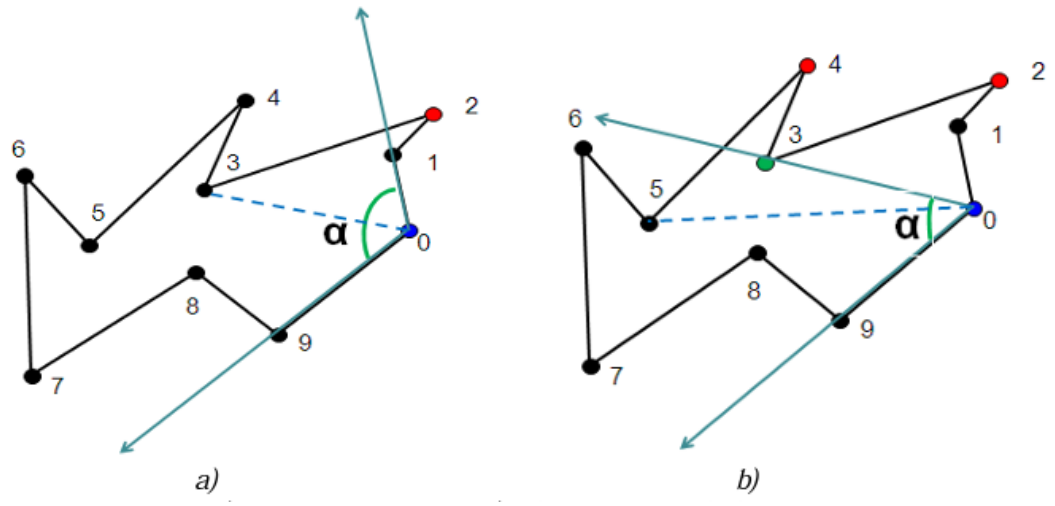


Abbildung 11: (a) v_3 ist sichtbar für v_0 (b) Überprüfung der Sichtbarkeit von v_5 mit neuem Sichtfeld α (nicht sichtbar entspricht rot, sichtbar entspricht grün) [9]

Für die Anzahl sichtbarer Punkte bezogen auf v_i wird die hier Bezeichnung $s(v_i)$ verwendet. Man kann diesen Vorgang der Sichtbarkeitsanalyse wie folgt in Pseudocode beschreiben.

Algorithmus 3: Sichtbarkeitsanalyse für einen Punkt v_i

Eingabe: Ecken $v_i \in V(P),\ n = |V(P)| - 3,\ s(v_i) = 0$
Ausgabe: Anzahl sichtbarer Punkte $s(v_i)$
Schritt 1: Wähle Referenzpunkt v_i
Schritt 2: $\alpha = \angle v_{i-1} v_i v_{i+1}$
Schritt 3: **while** $n > 0$ **do**
if $\angle v_{i-1} v_i v_j > 0 \wedge \angle v_{i-1} v_i v_j < \alpha$
if $v_i v_j$ Diagonale von P
(a) $s(v_i) = s(v_i) + 1$
(b) $\alpha = \angle v_{i-1} v_i v_j$
(c) $n = n - 1$
Schritt 4: Ausgabe von $s(v_i)$

Ist $s(v_i)$ für jeden Eckpunkt v_i von P ermittelt, wird der Punkt mit der größten solchen Anzhal ausgewählt. Gibt es mehrere solche Punkte, dann kann ein beliebige dieser Punkte gewählt werden. Dieser Punkt wird dann als **First Head** bezeichnet. Als nächstes wird noch der Punkt mit der zweit höchsten Anzahl $s(v_i)$ gewählt. Er wird als **Second Head** bezeichnet. Sollte es hier Uneindeutigkeiten bei der Auswahl geben, dann wird ein Test durchgeführt, um diesen Punkt auszuwählen. Durch die Diagonale zwischen First und Second Head soll das Polygon in zwei Unterpolygone zerlegt werden. Man testet, bei welchem der möglichen Punkte für den Second Head, der Betrag der Differenz zwischen der Anzahl an Strecken in

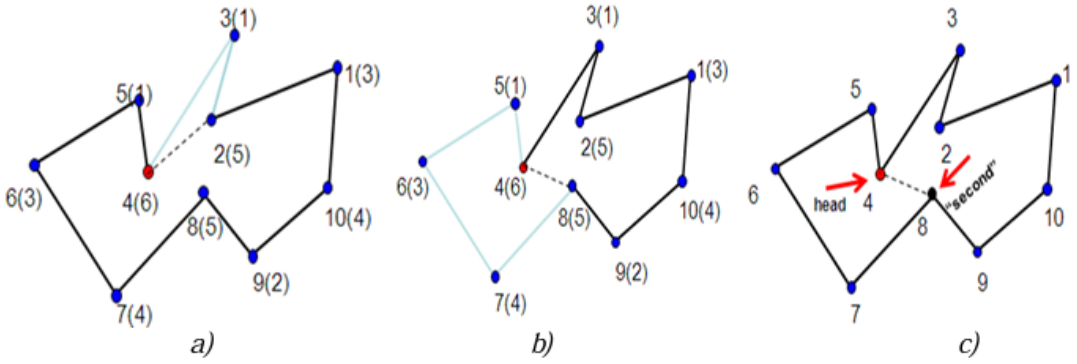


Abbildung 12: (a) und (b) Vergleich der Differenz zwischen der Länge des schwazen und des türkiesen Strecknabschnitts (c) Auswahl des Second Head [9]

den beiden Unterpolygonen P_1 und P_2 am geringsten ist. Das ist in der nächsten Abbildung zu sehen.

Für die Zerlegung werden First und Second Head dann dupliziert, damit beide Polygone vollständig begrenzt sind. In beiden Unterpolygonen muss dann die Sichtbarkeitsanalyse erneut durchgeführt werden. Damit dies ein wenig schneller geschieht, kann man die sichtbaren Punkte im Bezug auf einen Punkt v_i in einer sogennanten *single circular list* gespeichert werden. In einer solchen Liste zeigt der Pointer des letzten Elements auf das erste Element der Liste. Beim Update der Sichtbarkeiten müssen dann für jeden Punkt nur die Punkte in seiner jeweiligen Liste überprüft werden und jetzt nicht mehr sichtbare Punkte aus der Liste gelöscht werden. Damit wird die Laufzeit der Sichtbarkeitsanalyse von $O(n^2)$ auf $O(n)$ begrenzt. Nichtsdesdotrotz hat der neue Algorithmus von Ran Liu, nach seiner eigenen groben Analyse, eine Komplexität von über $O(n^3)$. Im Vergleich mit dem ECA, welcher eine Komplexität von $O(n^2)$ besitzt, schneidet er damit schlechter ab. In Tabelle 2 und 3 werden zwei Beispiele des Vergleichs zwischen dem Sichtbarkeitsalgorithmus und dem ECA aus der Arbeit von Liu angeführt. Die Algorithmen wurden auf zufällig generierten Polygone unterschiedlicher Knotenanzahl und Form getestet. Dabei waren die getesteten Polygonformen einmal *rund*, das heißt die Eckpunkte konnten nur in einem quadratischen Koordinatenbereich liegen. Der andere Typ war *länglich*, wobei die Punkte eher in ihrer x-Koordinate weiter streuten, nicht so sehr jedoch in der y-Koordinate. Es ist in den Ergebnissen dieser Tests ersichtlich, dass die Qualität der Dreicke bezogen auf ihre minimalen Innenwinkel bei Liu's Algorithmus besser ist, als die des ECA. In Sachen Laufzeit jedoch schneitet der neue Algorithmus jedoch schlechter ab, wie bereits die Komplexitätsanalyse zeigte.

In der nachfolgenden Abbildung ist zunächst einmal exemplarisch dargestellt, wie eine solche Zerlegung eines einfachen Polygons durchgeführt wird. Danach folgen die angesprochenen Tabellen.

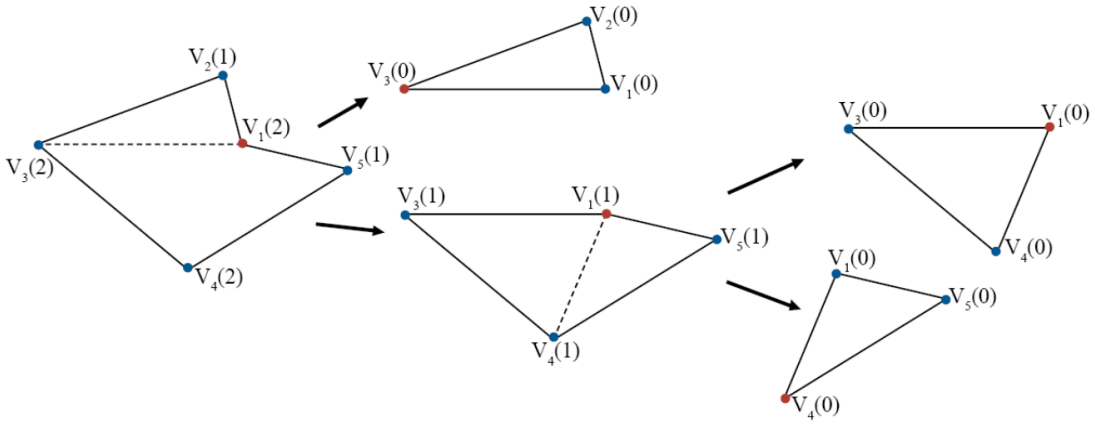


Abbildung 13: Von links nach rechts die Schritte der Unterteilung eines Polygons in Unterpolygone bis zur Triangulierung mittels Sichtbarkeit der Eckpunkte (rot die First Heads)[9]

Eckenanzahl: 30 $v_i = \{(x,y) \mid -50 < x < 50, -50 < y < 50\}$				
Algorithmus	Polygon-fläche	Durchschn. Dreiecksfläche	Standartabw. der Dreiecksfläche	Durchschn. min. Winkel
Ear-Clipping	4128,5px	147,446px	163,63px	10,29°
Sichtbarkeit	4128,5px	147,446px	148,91px	15,04°

Tabelle 2: Vergleich des ECA und des Sichtbarkeitsalgorithmus bei einem rundem Polygon mit 30 Ecken anhand der Standartabweichung der Dreiecksfläche in Pixeln (je kleiner desdo besser) und der durchschnittlichen minimalen Innenwinkelgröße in Grad (je größer desdo besser)[9]

Eckenanzahl: 30 $v_i = \{(x,y) \mid -100 < x < 100, -30 < y < 30\}$				
Algorithmus	Polygon-fläche	Durchschn. Dreiecksfläche	Standartabw. der Dreiecksfläche	Durchschn. min. Winkel
Ear-Clipping	5066px	180,93px	185,50px	9,23°
Sichtbarkeit	5066px	180,93px	146,03px	16,45°

Tabelle 3: Vergleich des ECA und des Sichtbarkeitsalgorithmus bei einem länglichen Polygon mit 30 Ecken anhand der Standartabweichung der Dreiecksfläche in Pixeln (je kleiner desdo besser) und der durchschnittlichen minimalen Innenwinkelgröße in Grad (je größer desdo besser) [9]

Literatur

[1] *Digitale Revolution*
(https://www.staatslexikon-online.de)

[2] *Darstellung von Kurven und Flächen*, Christoph Dähne
(https://www.inf.tu-dresden.de)

[3] *Geometrical Mesh Quality*
(https://www.iue.tuwien.ac.at)

[4] *Sliver In: Computer Graphics Dictionary*, Stevens, R.T., 2002.

[5] *Triangulation by Ear Clipping*, David Eberly
(https://www.geometrictools.com)

[6] *Polygon Triangulation*, Subhash Suri
(https://sites.cs.ucsb.edu/ suri/cs235/Triangulation.pdf)

[7] *Parallelized ear clipping for the triangulation and constrained Delaunay triangulation of polygons*, Günther Eder, Martin Held, Peter Palfrader
(https://www.sciencedirect.com)

[8] *Improved Algoritms For Ear-CLipping Triangulation*, Bartosz Kajak, 2005
(https://digitalscholarship.unlv.edu/thesesdissertations/1319/)

[9] *A comparison of Ear Clipping and a new Polygon Triangulation Algorithm*, Ran Liu
(https://www.diva-portal.org/smash/get/diva2:330344/FULLTEXT02.pdf)

[10] *Polygon, Definition*
(https://mathepedia.de/Polygone.html)

[11] *Regular Polygons. In: Michiel Hazewinkel (Hrsg.): Encyclopedia of Mathematics. Springer-Verlag und EMS Press, Berlin 2002*

[12] *Convex Polygon*
(https://www.mathopenref.com/polygonconvex.html)

[13] *Triangulation, Definition*
(https://encyclopediaofmath.org/wiki/Triangulation)

[14] *Simplex, Definition*
(https://encyclopediaofmath.org/wiki/Simplex)

[15] *Ear-clipping Based Algorithms of Generating High-quality Polygon Triangulation*, Gang Mei, John C.Tipper and Nengxiong Xu
(https://arxiv.org/ftp/arxiv/papers/1212/1212.6038.pdf)

[16] *Ear-Clipping Triangulierung*
(wiki.delphigl.com)

[17] *Jordanscher Kurvensatz*
(https://de.wikipedia.org)

[18] *The smallest 8 cubes to cover a regular tetrahedron* (https://math.stackexchange.com/)

[19] *Slicing an ear using prune-and-search In: Pattern Recognition Letters*, ElGindy, H., Everett, H., and Toussaint, G. T., (1993) S. 719-722

[20] *Polygons Have Ears In: Amer. Math. Monthly*, G.H. Meisters, Ausgabe 82, S. 648–651, 1975.

[21] *Turing-Maschinen In: Der Turing Omnibus*, A. K. Dewdney, S. 211-230, 1975.

[22] *Computational Geometry In: C. Cambridge: Cambridge University Press*, O'Rourke, J., (1998).

[23] *Triangulating a polygon in parallel In: Journal of Algorithms*, M. Goodrich, Ausgabe 10, S. 327-351, 1989. (https://www.sciencedirect.com/)

[24] *FIST: Fast Industrial-Strength Triangulation of Polygons In: Algorithmica*, Held, M., Ausgabe 30, S. 563–596, 2001. ()https://link.springer.com/article/10.1007/s00453-001-0028-4)

[25] *Reentrant Polygon Clipping*, Sutherland, Ivan E. und Hodgman, Gary W., 1974. (https://dl.acm.org/doi/abs/10.1145/360767.360802)