Seminar Computergrafik
Universität Göttingen

Ausarbeitung

3D Rasterung
Eine Vereinigung von Rasterung and Ray Casting

Ausgearbeitet von: Simon Christoph Stein
E-Mail: s.stein1@stud.uni-goettingen.de

Dozenten: Prof. Dr. Winfried Kurth
Dipl.-Inf. Reinhard Hemmerling

Inhaltsverzeichnis

1 Einleitung **3**

2 Traditionelle Renderverfahren **4**
2.1 Rasterung 4
2.2 Raytracing 7

3 3D-Rasterung **9**
3.1 Algorithmische Funktionsweise 9
3.2 Vergleich 2D/3D-Rasterung 12

4 Anwendung von 3D-Rasterung **13**
4.1 Nicht-planare Viewports 13
4.2 Kantenglättung und eindeutiges Rendern 15
4.3 Kontinuierlich veränderbare Traversierung von Szenen 16

5 Zusammenfassung **17**

6 Quellen **18**

1 Einleitung

Die Computergrafik hat seit ihrem Aufkommen in den frühen 1950er Jahren einen weiten Weg zurückgelegt [3, 4]. Mit wenigen primitiven geometrischen Objekten auf den Monitoren einiger Großrechner begann eine rasend schnelle Entwicklung. Deren Weg führte über die breite Verfügbarkeit von Computergrafikanwendungen für Privatanwender, bis hin zu den fast fotorealistischen computergenerierten Effekten moderner Kinofilme, wie wir sie heute kennen. Schon in einem frühen Stadium dieser Entwicklung kristallisierten sich die beiden wichtigsten Algorithmen heraus, welche bis heute zur Erzeugung computergenerierter Bilder verwendet werden: `Rasterung` [6] und `Raytracing` [1, 8].

Aufgrund des geringen Berechnungsaufwands und der leichten Umsetzbarkeit des Algorithmus in Hardware setzte sich Rasterung als der Standard für die meisten Anwendungen, insbesondere im privaten Bereich, durch. Dies ist vor allem darauf zurückzuführen, dass bei Rasterung ein Großteil der Arbeit mit Festkommazahlen durchgeführt werden kann, während Raytracing Gleitkommaberechnungen benötigt. Aus konzeptionellen Gründen stößt Rasterung allerdings bei der Berechnung globaler Effekte und physikalisch korrekter Beleuchtung auf große Schwierigkeiten, was sich in der Entwicklung zu immer wirklichkeitsgetreueren Computergrafiken als Stolperstein erweist.

Raytracing wird heutzutage vorallem im professionellen Bereich, zum Beispiel für CAD-Anwendungen und bei der Berechnung von Effekten in Kinofilmen, angewendet. Der Algorithmus eignet sich prinzipstechnisch sehr gut für die Berechnung realitätsnaher Bilder mit physikalisch korrekter Beleuchtung, fordert dafür allerdings viel Performance ein. Da sich die Entwicklung von Grafikhardware, anders als bei Rasterung, als schwierig erwies und eine Softwarelösung als nicht kompetitiv in Echtzeitanwendungen eingeschätzt wurde, kam die Entwicklung von Raytracing während der 1990er Jahre fast zum Stillstand. In der letzten Zeit wird das Interesse an Raytracing aufgrund der gestiegenen Rechenleistung und der Entwicklung hin zu Many-Core-Prozessoren mit hoher Fließkomma-Leistung wieder größer [7].

In dieser Ausarbeitung wird ein Algorithmus vorgestellt, welcher Eigenschaften von Rasterung und Raytracing vereint und in der Lage ist, Konzepte zwischen beiden Verfahren zu übertragen. Das „3D-Rasterung“ genannte Verfahren wurde 2009 von Dachsbacher *et al.* vorgestellt [5]. Nach einem kurzen Blick auf die herkömmlichen Renderverfahren wird die Funktionsweise des Algorithmus erläutert und auf die sich daraus ergebenden Möglichkeiten eingegangen.

2 Traditionelle Renderverfahren

Im Folgenden werden die üblichen Renderverfahren `Rasterung` und `Raytracing` kurz vorgestellt und ihre grundlegende Funktionsweise erläutert. Aufbauend auf den hier beschriebenen Konzepten wird in Abschnitt 3 dann das Verfahren der `3D-Rasterung` eingeführt.

2.1 Rasterung

Rasterung ist der derzeitige Standard für Echzeitgrafik im Bereich der Privatanwender. Nahezu alle aktuelle Grafikhardware ist auf dieses Verfahren ausgelegt und gerade im Bereich der Unterhaltungssoftware wird praktisch nichts anderes verwendet. Die populärsten Programmierschnittstellen zur Grafikprogrammierung unter Verwendung von Rasterung sind `OpenGL` und `DirectX` (bzw. `Direct3D`).

Die typische Renderingpipeline für Rasterung umfasst in etwa folgende Schritte [5]:

1. Modeling & Viewing Transformation (optional: Beleuchtung)
2. Clipping & Projektion
3. (Hierachische) Rasterung & Early Z-Test, Interpolation von Vertex-Attributen, Pixel-Shading, Texturierung
4. Tiefentest, Stencil Operationen, Blending etc.

Die eigentliche Rasterung findet hier nur im dritten Schritt statt. Nachdem die Vertices in ein gewünschtes Koordinatensystem transformiert und Vertices außerhalb des Sichtbereichs durch Clipping entfernt wurden, werden die 3D-Objekte in den zweidimensionalen Bildraum projiziert. Von hier aus müssen die bisher noch kontinuierlichen Objekte den diskreten Pixeln des Bildschirms zugeordnet werden. Dieser Schritt wird als Rasterung bezeichnet. Für jedes Dreieck der Szene wird nun getestet, welche Pixel von ihm bedeckt werden. Diese Reihenfolge der Traversierung der Szene ist ein charakteristisches Merkmal des Algorithmus.

Traversierung der Szene bei Rasterung:

```
Für jedes Dreieck{
    Für jedes Pixel{
      Wenn(Sichtbar(Pixel,Dreieck))
      Ausgabe
    }
}
```

2D-Kantenfunktionen

Für die Prüfung ob ein Dreieck ein bestimmtes Pixel überdeckt, wird ein möglichst effizientes Verfahren benötigt. Eine übliche Methode zur Lösung dieses Problem ist die Verwendung von `2D-Kantenfunktionen` [8]. Eine solche Funktion hat die Eigenschaft, den Raum entlang einer Kante in zwei Bereiche aufzuteilen (Abb. 1).

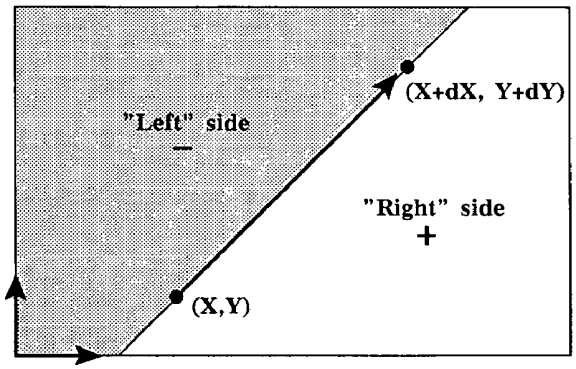

Abbildung 1: 2D-Kantenfunktion [8]

Gegeben sei die Kantenfunktion [8]:

$$E(x,y) = (x-X)dY - (y-Y)dX = \left(\begin{pmatrix} x-X \\ y-Y \\ 0 \end{pmatrix} \times \begin{pmatrix} (X+dX)-X \\ (Y+dY)-Y \\ 0 \end{pmatrix} \right) \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Fasst man die Verbindungslinie zwischen (X,Y) und $(X+dX, Y+dY)$ als Definition einer Kante auf, so erfüllt $E(x,y)$ folgende Eigenschaften:

- $E(x,y) > 0$ wenn (x,y) auf der „rechten Seite“ der Kante
- $E(x,y) < 0$ wenn (x,y) auf der „linken Seite“ der Kante
- $E(x,y) = 0$ wenn (x,y) auf der Kante

Über diese Kantenfunktion lässt sich feststellen, ob ein Punkt innerhalb eines Dreiecks liegt[1]. Sind die Vertices eines Dreiecks konsistent nummeriert (z.B. immer im mathematisch positiven Sinn), so ist ein Pixel innerhalb eines Dreiecks, wenn $E_i(x, y) > 0$ für alle Kantenfunktionen $i = 0, 1, 2$ gilt (Abb. 2, links).

Ein weiterer Vorteil der Kantenfunktion ist die Linearität in den Bildkoordinaten, aufgrund derer sich die Funktion inkrementell bestimmen lässt. Dies ermöglicht auch die unabhängige, parallele Abarbeitung unterschiedlicher Koordinaten. Es gilt:

- $E(x + 1, y) = E(x, y) + dY$
- $E(x, y + 1) = E(x, y) - dX$

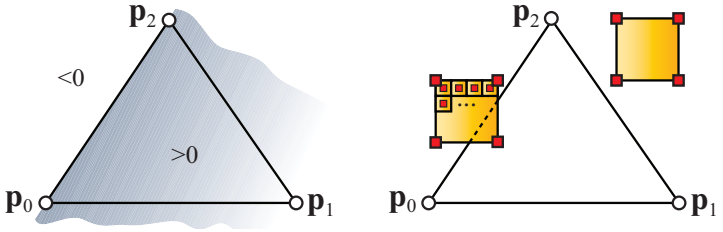

Abbildung 2: Überdeckungsbestimmung (links) und Binning (rechts) [5] .

Wichtig für eine hohe Geschwindigkeit ist **Binning**. Darunter versteht man eine schnelle Schätzung, welche Pixelbereiche für ein gegebenes Dreieck geprüft werden müssen, damit nicht stets die gesamte Bildfläche abgearbeitet werden muss. Üblich ist die Prüfung an den vier Eckpunkten eines quadratischen „Bins“ gegen ein gegebenes Dreieck (Abb. 2, rechts). Die in einem gegebenen Bin befindlichen Pixel werden nur gerastert, wenn sich auch Teile des Dreiecks in ihm befinden. Da für einen leeren Bin nur die vier Prüfungen an den Eckpunkte anfallen, lässt sich so eine Menge Performance einsparen.

Ein naiver Ansatz für Binning ist die Abarbeitung der `Bounding Box` eines Dreiecks. Unter einer Bounding Box versteht man das kleineste Rechteck, welches das Dreieck umschließt (Abb. 3, links).

[1] Anmerkung: Natürlich sind auch andere Definitionen der Kantenfunktion möglich. Zum Beispiel kann man das Skalarprodukt von einem Vektor mit dem Kreuzprodukt zwischen dem anderen Vektoren und dem z-Einheitsvektor bilden (was in 2D dem Tausch beider Komponten und einem Vorzeichenwechsel einer Komponte entspricht):
$$E^*(x, y) = \begin{pmatrix} x - X \\ y - Y \\ 0 \end{pmatrix} \cdot \left(\begin{pmatrix} dX \\ dY \\ 0 \end{pmatrix} \times \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \right) = \begin{pmatrix} x - X \\ y - Y \\ 0 \end{pmatrix} \cdot \begin{pmatrix} dY \\ -dX \\ 0 \end{pmatrix}$$
Bei der Berechnung im Computer kann es bei unterschiedlichen Definitionen zu Performanceunterschieden kommen.

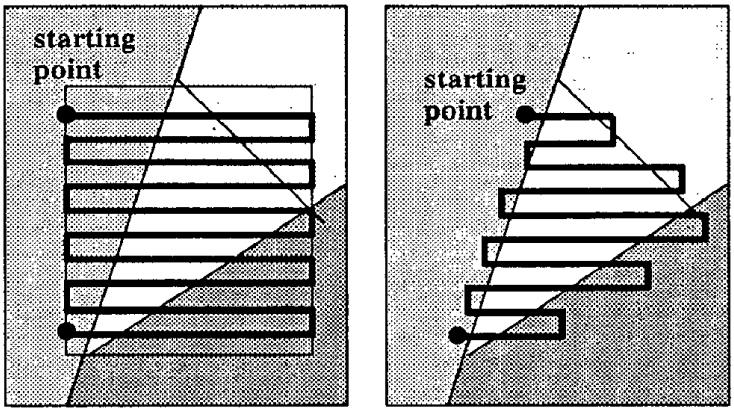

Abbildung 3: Abarbeitung der Pixel die durch eine Bounding Box (links) bzw. hierarchisches Binning (rechts) vorgegeben wurden [8].

Die Herangehensweise über eine Bounding Box ist bei langen und dünnen Dreiecken allerdings sehr ineffizient, da große Teile der Bounding Box nicht vom Dreieck überdeckt werden. Besser sind hier andere Binning-Verfahren, zum Beispiel über hierarchische Strukturen, welche zuerst große Bereiche prüfen und relevante Teile des Bildraums rekursiv immer weiter aufteilen (beispielsweise Quad Trees).

2.2 Raytracing

Raytracing kommt aktuell bevorzugt im professionellen Bereich (CAD, Kinofilme) zum Einsatz. Die Sichtbarkeitstests zur Entscheidung, welches Dreieck zu einem gegebenen Pixel gehört, finden hier durch Strahlverfolgung direkt in Welt- oder Kamerakoordinaten statt. Da die zugehörigen Berechnungen im 3D-Raum operieren, muss hier Fließkomma-Arithmetik verwendet werden. Für jeden Pixel wird ein Strahl vom Auge des Betrachters in die Szene geschickt und bestimmt, welche Dreiecke von ihm geschnitten werden (Abb. 4). Die Traversierung der Szene ist hier genau umgedreht gegenüber Rasterung, für ein gegebenes Pixel werden alle Dreiecke geprüft.

Traversierung der Szene bei Raytracing:

```
Für jedes Pixel{
    Für jedes Dreieck{
      Wenn(Sichtbar(Pixel,Dreieck))
      Ausgabe
    }
}
```

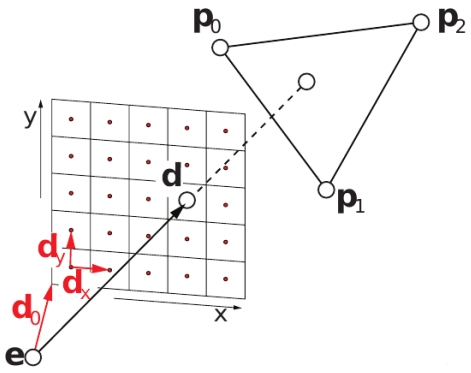

Abbildung 4: Aussendung von Primärstrahlen bei Raytracing (Ray Casting) [5].

Ähnlich dem Gedanken von Binning bei der Rasterung lassen sich auch hier Beschleunigungsstrukturen einsetzen, um nicht für jeden Strahl gegen jedes Dreieck testen zu müssen. Diese optimierten Raumstrukturen müssen im Fall von Raytracing allerdings direkt in 3D operieren. Dabei wird versucht, große Teile des Raumes, die der Strahl nicht durchquert, früh in der Berechnung zu verwerfen. Hohe Verbreitung haben hier vor allem `Octree`, `kd-tree`, `BVH` (*Bounding Volume Hierarchies*) oder `3D Gitter`. Eine wichtige Anmerkung ist, dass der Rechenaufwand bei Raytracing logarithmisch mit der Anzahl der Primitive (Rasterung: linear) und linear mit der Anzahl der Pixel steigt. Für sehr komplexe Szenen mit vielen Dreiecken ergibt sich hieraus ein Geschwindigkeitsvorteil für Raytracing. Ohne spezielle Hardware ist Rasterung bei heutigen Echtzeitanwendungen allerdings noch weit schneller.

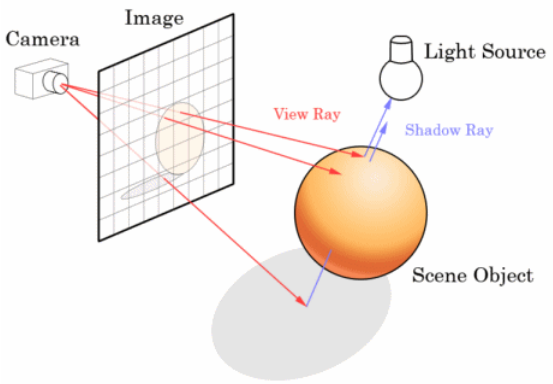

Abbildung 5: Beleuchtungsberechnung beim Raytracing [2].

Wird nur die Sichtbarkeitsprüfung über die Primärstrahlen verwendet, spricht man oft von **Ray Casting**. Erst wenn vom Auftreffpunkt weitere Sekundärstrahlen ausgesandt werden (z.B. reflektierter und gebrochener Strahl sowie Schattenstrahl zur Beleuchtungsrechnung, Abb. 5) verwendet man den Begriff **Raytracing**. Strahlbasierte Verfahren bieten aufgrund ihrer Konzeption ein gutes Framework für fotorealistische Beleuchtung und globale Effekte.

3 3D-Rasterung

In diesem Abschnitt wird das Verfahren der `3D-Rasterung` besprochen. Dieses wurde von DACHSBACHER *et al.* auf der Suche nach einem Algorithmus entworfen, der sowohl Eigenschaften von Raytracing als auch Rasterung besitzt. Auf diese Weise wird es möglich, Konzepte zwischen beiden Renderverfahren auszutauschen, welche in einem Verfahren besonders leicht zu realisieren sind, während sie im jeweils anderen gar nicht oder nur unter besonderem Aufwand verwendet werden können. Eine Auswahl der Möglichkeiten wird in Abschnitt 4 vorgestellt. 3D-Rasterung basiert auf der Idee, dass der Test, ob ein Pixel von einem 2D-Dreieck bedeckt wird, äquivalent dazu ist, ob ein Strahl vom Beobachter durch diesen Pixel das Dreieck schneidet. Das Verfahren verwendet daher eine Ray-Casting-Variante und arbeitet mit Fließkommaoperationen direkt in Welt- oder Kamerakoordinaten, ähnlich zu Raytracing. Aus diesem Grund ist es für die Verwendung auf hochparallelisierten Architekturen wie Intel's *Knights Ferry* (vormals *Larrabee*) gedacht, welche eine besonders hohe Performance bei der Verarbeitung paralleler Fließkommaoperationen bieten. Trotz seiner Verwandschaft zu Raytracing auf der einen Seite, ähneln die Grundgedanken bei der Verarbeitung und Traversierung der Szene aber eher traditioneller 2D-Rasterung.

3.1 Algorithmische Funktionsweise

Ähnlich zur 2D-Rasterung wollen wir auch hier eine Kantenfunktion definieren. Über die `3D-Kantenfunktion` aller Kanten eines gegebenen Dreiecks soll bestimmbar sein, ob der Strahl von Auge des Beobachters durch ein bestimmtes Pixel dieses Dreieck schneidet. Um die Rechnung zu vereinfachen nehmen wir an, dass sich der Beobachter im Ursprung befindet ($\vec{e} = 0$)[2]. Unser Sichtfenster (*Viewport*) und damit auch die Richtung aller benötigten Strahlen ist dann durch $\vec{d} = \vec{d_0} + x\vec{d_x} + y\vec{d_y}$ gegeben. Wir betrachten ein Dreieck mit den Vertices $\vec{p}_0, \vec{p}_1, \vec{p}_2$, Abbildung 6 illustriert die Situation.

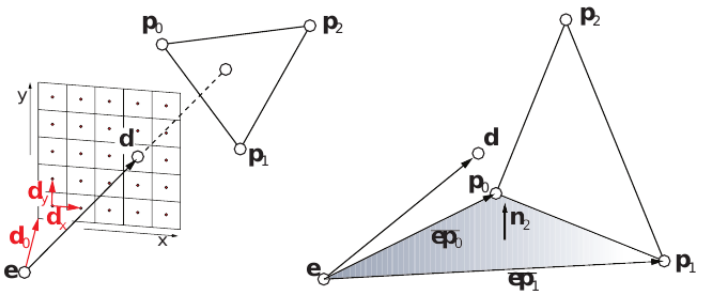

Abbildung 6: Ausgangssituation bei 3D-Rasterung [5].

[2]Das Verfahren arbeitet natürlich auch problemlos mit $\vec{e} \neq 0$.

Im Folgenden wollen wir uns die Kantenfunktion für die Kante $\overline{p_0 p_1}$ überlegen. Dazu benötigen wir den Normalenvektor, welcher auf der von $\vec{e}$, $\vec{p}_0$ und $\vec{p}_1$ aufgespannten Ebene steht. Es ist $\vec{n}_2 = \vec{p}_1 \times \vec{p}_0$ (Abb. 6, rechts).

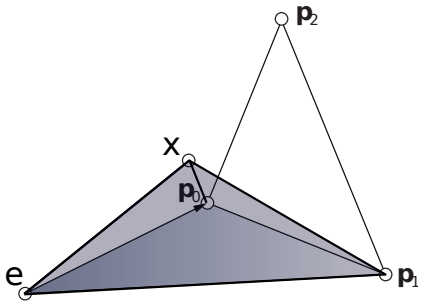

Abbildung 7: Tetraeder zwischen $\vec{p}_0, \vec{p}_1, \vec{x}$ und $\vec{e}$.

Das Spatprodukt $V_2(\vec{x}) = \vec{n}_2 \cdot \vec{x} = (\vec{p}_1 \times \vec{p}_0) \cdot \vec{x}$ liefert das Volumen des Tetraeders, der von $\vec{p}_0, \vec{p}_1, \vec{x}$ und $\vec{e}$ aufgespannt wird (Abb. 7). Der Skalenfaktor von $\frac{1}{6}$ gegenüber dem Volumen des aufgespannten Parallelepipeds wurde hier weggelassen, da die Methode nur mit Vorzeichen und Verhältnissen arbeitet und die Funktionsweise anhand der Tetraeder leichter zu veranschaulichen ist. Für einen beliebigen Vektor $\vec{x}$ gilt:

- $V_2(\vec{x}) > 0$ für alle $\vec{x}$ im Halbraum der $\vec{p}_2$ enthält.
- $V_2(\vec{x}) < 0$ für alle $\vec{x}$ im Halbraum der $\vec{p}_2$ nicht enthält.
- $V_2(\vec{x}) = 0$ für alle $\vec{x}$ mit $\vec{x} \perp \vec{n}_2$.

Die Verwandschaft zur 2D-Kantenfunktion ist hier deutlich zu erkennen. Nun definieren wir etwas allgemeiner:

$$\vec{n}_i := \vec{p}_{(i+2) \mod 3} \times \vec{p}_{(i+1) \mod 3}$$

$$V_i(\vec{x}) := \vec{n}_i \cdot \vec{x} \text{ mit } i = 0, 1, 2$$

Das Gesamtvolumen des Tetraeders, der vom Dreieck mit dem Ursprung aufgespannt wird ist dann $V := \vec{p}_j \cdot \vec{n}_j$ für $j = 0, 1, 2$ beliebig (Spatprodukt aller Vertices). Anhand dieser Definitionen können wir nun folgende Aussagen treffen:

- Der Strahl $\vec{e} + t'\vec{d},\ t' > 0$ trifft das Dreieck, wenn alle vier Volumina das gleiche Vorzeichen haben.
- $V_i, V > 0\ \forall i$, Strahl trifft auf Frontfläche des Dreiecks
- $V_i, V < 0\ \forall i$, Strahl trifft auf Rückseite des Dreiecks
- $V_i, V = 0\ \forall i$, Position des Beobachters $\vec{e}$ liegt in Ebene des Dreiecks

Am Schnittpunkt vom Strahl mit dem Dreieck ergibt die Addition der Volumina V_i aller Kanten das Gesamtvolumen V (Abb. 8). Aus dieser Beziehung können wir den Strahlparameter t für den Schnittpunkt gewinnen:

$$V = V_0(t\vec{d}) + V_1(t\vec{d}) + V_2(t\vec{d})$$
$$t = V/(V_0(\vec{d}) + V_1(\vec{d}) + V_2(\vec{d}))$$

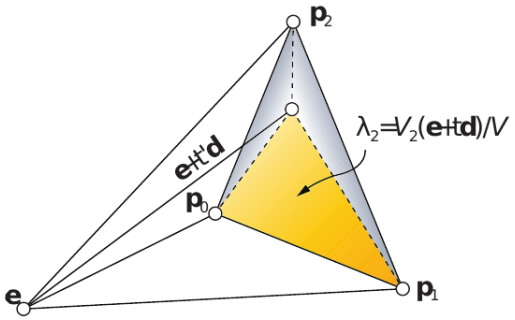

Abbildung 8: 3D Kantenfunktionen am Schnittpunkt [5].

Auch die Baryzentrischen Koordinaten des Schnittpunkts lassen sich mit den Verhältnissen der Volumina der Kanten zum Gesamtvolumen leicht bestimmen. Am Schnittpunkt gilt für die Baryzentrischen Koordinaten:

$$t\vec{d} = \lambda_0\vec{p}_0 + \lambda_1\vec{p}_1 + \lambda_2\vec{p}_2$$
$$\text{mit } \lambda_i = V_i(\vec{d})/(V_0(\vec{d}) + V_1(\vec{d}) + V_2(\vec{d}))$$

Der Wert von λ_2 ist in Abbildung 8 grafisch dargestellt. Es ist anzumerken, dass der Strahlparameter t weder für den Schnitttest noch die Berechnung der Baryzentrischen Koordinaten benötigt wird. Allerdings kann mit t der Schnittpunkt direkt in Welt- bzw. Kamerakoordinaten berechnet werden. Desweiteren lässt sich der Wert des Strahlparameters auch für einen Tiefentest verwenden, da das geschnittene Dreieck mit dem kleinsten Wert von t am nächsten am Beobachter ist (ähnlich dem Z-Buffer bei 2D-Rasterung).

Als weiteres nützliches Werkzeug können wir die räumliche Ableitung der Kantenvolumina definieren (hier für einen planaren Viewport):

$$V_i(\vec{d}) = \vec{n}_i \cdot \vec{d} = \vec{n}_i \cdot (\vec{d}_0 + x\vec{d}_x + y\vec{d}_y)$$
$$\Rightarrow V_{i,x} = \frac{\partial V_i(\vec{d})}{\partial x} = \vec{n}_i \cdot \vec{d}_x, \qquad V_{i,y} = \frac{\partial V_i(\vec{d})}{\partial y} = \vec{n}_i \cdot \vec{d}_y$$

Im Fall eines planaren Viewports lässt sich die Überdeckungsprüfung mithilfe der Ableitungen genauso effizient berechnen wie bei 2D-Rasterung, denn für ein gegebenes Pixel (x, y) ist für jede Kante nur die Berechnung von

$$V_i(\vec{d}_0) + xV_{i,x} + yV_{i,y}$$

nötig. Auch bei komplexeren Viewports lässt sich ein (gegebenenfalls mehrstufiges) inkrementelles Berechnungsschema finden. Die Ableitungen lassen sich auch dazu verwenden, Pixel in Grenzfällen (z.B: Pixelzentrum genau auf einem Vertex) eindeutig einem Dreieck zuzuordnen (siehe Abschnitt 4.2).

3.2 Vergleich 2D/3D-Rasterung

In diesem Abschnitt sind einige Gemeinsamkeiten und Unterschiede von 3D-Rasterung zu herkömmlicher hierarchischer 2D-Rasterung aufgeführt:

- Beide nutzen Kantenfunktionen zur Sichtbarkeitsprüfung, welche entweder direkt oder inkrementell ausgewertet werden können.

- 3D-Rasterung kann schneller sein als 2D-Rasterung, insbesondere bei Berechnungen von perspektivisch korrekten Baryzentrischen Koordinaten, wo sie deutlich weniger Operationen benötigt.

- 3D-Rasterung erfordert keine Projektion und kein Clipping, wodurch auch nichtplanare Viewports einfach möglich werden, etwa um Objektivverzerrungen (z.B. Weitwinkel) darzustellen. In 2D-Rasterung sind die erforderlichen nichtlinearen Projektionen nur schwer durchzuführen.

- Beide Algorithmen können „Binning“ zur Geschwindigkeitssteigerung verwenden. Für 2D-Rasterung werden quadratische Bins im zweidimensionalen Bildraum verwendet, bei 3D-Rasterung entspricht dies der Aussendung eines Pyramidenstumpfs von Strahlen. 3D-Rasterung kann aber auch auf für Raytracing übliche Strukturen wie BVHs, kd-trees etc. zurückgreifen.

- 2D-Rasterung arbeitet mit Festkomma-Arithmetik, während 3D-Rasterung, ebenso wie Raytracing, Fließkomma-Arithmetik benötigt.

- 3D-Rasterung baut auf der für 2D-Rasterung üblichen Renderingpipeline auf, einige Schritte werden jedoch nicht mehr benötigt (z.B. Clipping & Projektion) oder können zusammengefasst werden. So ist es denkbar, keine Viewing Transformation mehr durchzuführen, sondern mit einer Raytracing-Kamera direkt in Weltkoordinaten zu rendern. Die herkömmliche Renderingpipeline kann verkürzt werden.

4 Anwendung von 3D-Rasterung

Es folgen einige Beispiele für Anwendungen von 3D-Rasterung. Der Algorithmus macht es möglich, Vorteile von herkömmlicher Rasterung und Raytracing gleichzeitig nutzen zu können.

4.1 Nicht-planare Viewports

Während nichtlineare Verzerrungen bei Rasterung nur schwierig darzustellen sind, sind sie in Raytracing leicht realisierbar. Aufgrund der in Abschnitt 3.1 vorgestellten Funktionsweise gilt letzteres auch für 3D-Rasterung. Eine Linsenverzerrung lässt sich zum Beispiel leicht realisieren, indem das Sichtfenster (der *Viewport*) verzerrt wird.

Beispiel: Bei der Verwendung einer parabolischen Funktion $f(x, y) = \frac{1}{2} - \frac{1}{2} \cdot (x^2 + y^2)$ für den Fenstervektor $\vec{d} = (x, y, f(x, y))^T$ ergibt sich ein Weitwinkeleffekt. Abbildung 9 zeigt den verzerrten Viewport durch den die Strahlen nun gesendet werden. Wie man leicht erkennen kann, werden die Strahlen dabei über einen größeren Winkelbereich verteilt, als es bei einem planaren Viewport der Fall wäre. Bei der Darstellung auf dem Bildschirm wird das Sichtfenster wieder „gerade gebogen“ und Teile außerhalb des üblichen Sichtkegels werden sichtbar. Das Ergebnis ist in Abbildung 10 (linkes Bild) dargestellt.

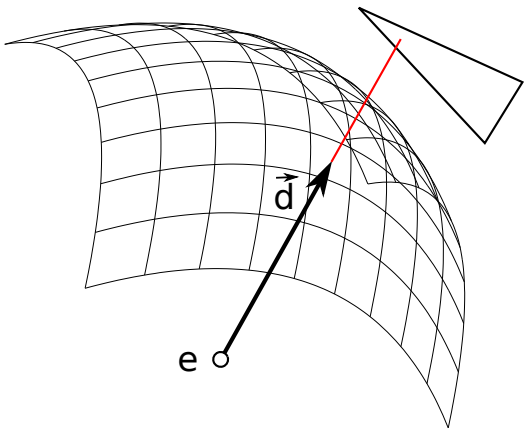

Abbildung 9: Parabolischer Viewport bei 3D-Rasterung.

Die Sichtbarkeitstests können direkt oder über ein zweistufiges inkrementelles Verfahren berechnet werden. Für eine inkrementelle Berechnung wird mit dem Pixel (x_0, y_0) und

dem dazugehörigen Richtungsvektor $\vec{d_0}$ gestartet. Es werden die Variablen $V = V_i(\vec{d_0})$, $V_x = V_{i,x}(\vec{d_0})$ und $V_y = V_{i,y}(\vec{d_0})$ initialisiert.

Für einen Sprung vom Pixel (x, y) zum Pixel $(x+\Delta x, y+\Delta y)$ muss folgende Berechnung durchgeführt werden (mit n_x, n_y, n_z den Komponenten des der Kante i zugehörigen Normalenvektors, siehe Abschnitt 3.1) [5]:

$$V \leftarrow V + V_i(\Delta x, \Delta y) - \frac{n_z}{2} + \Delta x(V_x - n_z) + \Delta y(V_y - n_z)$$
$$V_x \leftarrow V_x - n_x \Delta x, \qquad V_y \leftarrow V_y - n_y \Delta y$$

Abbildung 10: Nichtlineare Verzerrungen mit 3D-Rasterung [5].

Soll Binning verwendet werden, muss nun allerdings beachtet werden, dass der ursprüngliche Pyramidenstumpf zusammen mit dem Sichtfenster verzerrt wurde. Da nur an den Ecken eines Bins gegen ein gegebenes Dreieck geprüft wird, kann es nun passieren, dass ein Dreieck, welches sich eigentlich in dem verzerrten Stumpf befindet, nicht korrekt detektiert wird (Abb. 11, (a)).

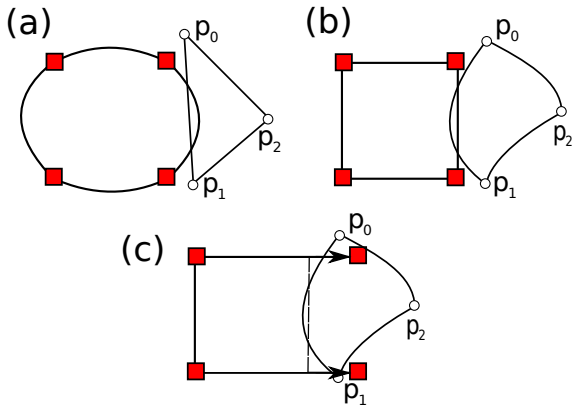

Abbildung 11: Verzerrter Bin aus der Sicht von Weltkoordinaten (a) und in der Bildebene (b). Virtuelle Verschiebung der Ecken für korrekte Erkennung des Dreiecks (c).

Das Problem lässt sich lösen, indem die dem Dreieck zugewandten Ecken des Bins virtuell verschoben werden (Abb. 11, (c)). Da die parabolische Projektion einer Linie ein Kreisbogen ist, erhält man die nötige Verschiebung aus der maximalen Eindringtiefe, die der zugehörige Kreis in den Bin haben kann, ohne einen der Eckpunkte zu schneiden. Falls der ermittelte Kreisradius kleiner als die halbe Breite des Bins ist, kann das Dreieck vollständig innerhalb des Bins liegen, ohne detektiert zu werden, weshalb der Bin in diesem Fall rekursiv geteilt und erneut getestet werden muss.

Ähnliche Strategien kann man für einige andere nichtlineare Projektionen auf ähnlichem Weg durch geometrische Konstruktion gewinnen.

4.2 Kantenglättung und eindeutiges Rendern

Während nichtlineare Projektionen in Raytracing einfacher umzusetzen sind, fallen *Anti-aliasing* (Kantenglättung) und eindeutiges Rendern eher in die Domäne der Rasterung. Im Algorithmus der 3D-Rasterung können die bestehenden Konzepte einfach genutzt werden.

Zur Kantenglättung lassen sich die bereits für herkömmliche 2D-Rasterung bekannten Methoden ohne große Modifikationen verwenden. In der Arbeit von Dachsbacher *et al.* [5] werden Multi-Sampling (MSAA) und Coverage-Sample-Anti-Aliasing (CSAA) eingesetzt (Abb. 12).

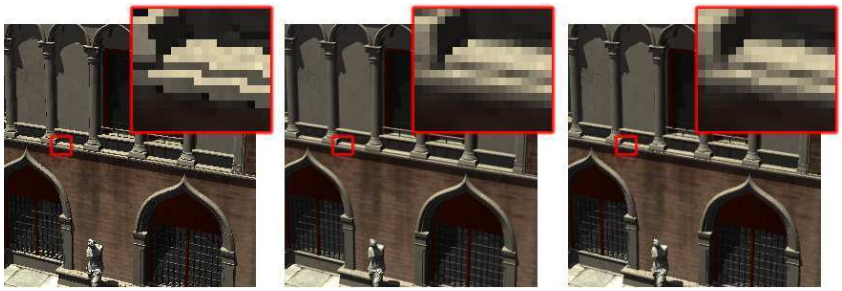

Abbildung 12: Bild ohne Kantenglättung (links), mit MSAA (mitte) und CSAA (rechts) bei 3D-Rasterung [5].

Bei 2D Rasterung existieren Konsistenzregeln, um ein Pixel das von mehreren Dreiecken geschnitten wird (d.h. es liegt auf einer Kante zweier benachbarter Dreiecke) eindeutig zuzuordnen. Üblich ist hier die *Top-Left-Convention*: Liegt ein Pixel[3] auf einer Kante, so gehört es zum Dreieck zu seiner Rechten. Wenn die Kante horizontal ist, gehört es zum

[3]Gemeint ist das Zentrum des Pixels, es sind aber auch andere Konventionen möglich.

Dreieck unter ihm. Durch diese einfachen Regeln wird jedes Pixel nur einmal gerendert, was Bildfehler (vorallem bei Blending und Transparenzen) verhindert.

Mithilfe der am Ende von Abschnitt 3.1 definierten Ableitungen $V_{i,x}$ und $V_{i,y}$ lassen sich auch für 3D-Rasterung solche Konsistenzregeln finden. Liegt ein Pixel auf einer Kante, so gilt ($V_{i,x} > 0$) nur für eines der beiden angrenzenden Dreiecke und diesem wird das Pixel zugeordnet. Ist die Kante horizontal ($V_{i,x} = 0$) wird die Zugehörigkeit nach dem sekundären Kriterium ($V_{i,y} < 0$) entschieden. Liegt ein Pixel auf einem Vertex, so werden für jedes angrenzende Dreieck die beiden in den Vertex laufenden Kanten i, j überprüft. Der Pixel ist dann dem Dreieck zugehörig, für das $(V_{i,x} > 0 \wedge V_{j,x} > 0) \vee (V_{i,x} = 0 \wedge V_{j,x} > 0)$ gilt.

Für Raytracing ist bisher kein vollständig konsistenter Renderansatz bekannt. Üblicherweise werden die Schnittpunktstest anhand der Baryzentrischen Koordinaten durchgeführt, was an Kanten und Ecken zu Mehrdeutigkeiten führt. Mit 3D-Rasterung lassen sich die Konsistenzregeln in natürlicher Weise auf Raytracing übertragen.

4.3 Kontinuierlich veränderbare Traversierung von Szenen

Wie schon in Abschnitt 2 dargestellt, ist einer der grundsätzlichen Unterschiede zwischen Rasterung und Raytracing die Reihenfolge, in der die Szene beim Rendern durchlaufen wird. Während Rasterung alle Dreiecke der Szene in beliebiger Reihenfolge bearbeitet und für ein gegebenes Dreieck auf Überdeckung mit allen Pixeln überprüft, wird bei Raytracing versucht, nur die minimale nötige Anzahl an Dreiecken auf dem Weg des Strahls eines gegebenen Pixels zu bearbeiten.

Ein bekannter Mittelweg ist, nur alle Dreiecke in einem vorgegebenen Pyramidenstumpf zu bearbeiten. Die Größe des Pyramidenstumpfes kann dabei den ganzen Bildschirm, nur Teile davon oder auch nur ein einzelnes Pixel umfassen. Eine Möglichkeit beim Rendering mit 3D-Rasterung wäre, den gesamten Sichtbereich aus mehreren Pyramidenstümpfen zusammenzusetzen, welche jeweils einzeln in herkömmlicher Weise gerastert werden. Gibt man den gesamten Bildschirm als Pyramidenstumpf vor, so entspricht dies der Vorgehensweise bei herkömmlicher 2D-Rasterung. Betrachtet man jedes Pixel für sich und führt jeweils Strahl-Dreieck-Schnittpunktstests durch, verhält sich der Algorithmus wie Raytracing.

Durch Variation der Größe der Pyramidenstümpfe zwischen diesen Extrema lässt sich die Traversierung kontinuierlich verändern. Zusätzlich zur Möglichkeit innerhalb der Stümpfe herkömmlich zu rastern, unterstützt 3D-Rasterung aber auch bei Raytracing übliche Methoden wie zum Beispiel Frustum- und Occlusion-Culling unter Nutzung einer BVH. Außerdem ist es möglich, gegebene Pyramidenstümpfe rekursiv zu unterteilen

(mit folgender Rasterung für Dreiecke innerhalb des Stumpfes oder Per-Pixel-Tests), wobei dann mit dem ganzen Bildschirm gestartet werden kann. Alle Rendervarianten sind zusätzlich sowohl mit als auch ohne Binning möglich.

5 Zusammenfassung

Mit 3D-Rasterung haben Dachsbacher *et al.* eine Rendermethode vorgestellt, welche sich im Kontinuum zwischen 2D-Rasterung und Ray Casting bewegt. Es wurde gezeigt, das sich mit 3D-Rasterung Konzepte zwischen Ray Casting und Rasterung erfolgreich austauschen lassen. Der Algorithmus bietet hierdurch vor allem viele neue Möglichkeiten beim Rendering und macht es möglich, gleichzeitig auf Verfahren zurückzugreifen, welche sonst nur schwer vereinbar sind. Dabei kann 3D-Rasterung bezogen auf die Anzahl der elementaren Operationen sogar merklich schneller sein als herkömmliche 2D-Rasterung.

Ein möglicher Vorteil der Technik in der Zukunft könnte sein, dass sie die Berechnung von globaler Beleuchtung (bzw. globalen Effekten) verbessern kann, welche in 2D-Rasterung sehr schwer durchzuführen ist und bei Raytracing einen hohen Rechenaufwand erzeugt. Dies könnte zu einer generellen Steigerung der Bildqualität von Echtzeitanwendungen beitragen. Desweiteren hoffen die Autoren von [3], die aktuell übliche Rendering-Pipeline zu vereinfachen.

Eines der größten Probleme des Verfahrens dürfte aktuell sein, dass genau wie bei Raytracing Fließkomma- statt Festkomma-Arithmetik eingesetzt werden muss. Die nötige Hardware, wie die in [3] angesprochene *Knights Ferry*-Karte von Intel, befindet sich zum jetzigen Zeitpunkt noch in der Entwicklung und ist für Privatanwender nicht erhältlich. In den nächsten Jahren könnte sich die Situation allerdings ändern und es ist bereits zu beobachten, dass die Industrie sich immer mehr den strahlbasierten Verfahren zuwendet. Die Ende 2010 vorgestellte `CryEngine 3` des deutschen Entwicklers `Crytek` mischt die herkömmliche Rasterung beispielsweise bereits mit einfachem Raytracing zur Berechnung lokaler Reflexionen. In Zukunft könnte der, trotz seiner Einfachheit sehr vielseitige, Renderansatz der 3D-Rasterung daher möglicherweise an Bedeutung gewinnen.

6 Quellen

1. A. Appel. *Some techniques for shading machine randerings of solids.* AFIPS '68 (Spring): Proceedings of the April 30-May 2, 1968, spring joint computer conference. Seiten: 37-45.

2. J. Atwood, *Real Time Raytracing*, 10. März 2008, `http://www.codinghorror.com/blog/2008/03/real-time-raytracing.html`
   - Letzter Zugriff: 28.12.2011 - 15:00 Uhr -

3. W. Carlson, *A Critical History of Computer Graphics and Animation*, The Ohio State University, 2003, `http://accad.osu.edu/~waynec/history/lessons.html`
   - Letzter Zugriff: 27.12.2011 - 18:00 Uhr -

4. S. H. Chasen, *Historical Highlights of Interactive Computer Graphics.*, Mechanical Engineering 103, 11, Nov. 1981. Seiten: 32-41

5. C. Dachsbacher, P. Slusallek, T. Davidovic, T. Engelhardt, M. Philips and I. Georgiev. *3D Rasterization - Unifying Rasterization and Ray Casting.* Technical report, Universität Stuttgart, 2009

6. J. Pineda. *A parallel algorithm for polygon rasterization.* Computer Graphics (Proceedings of SIGGRAPH '88), 22(4), 1988. Seiten: 17-20

7. D. Pohl, *Raytracing in Spielen*, Artikelserie, 2007-2011, `http://www.computerbase.de/artikel/grafikkarten/2011/bericht-ray-tracing-4.0/`
   - Letzter Zugriff: 05.01.2012 - 10:00 Uhr -

8. T. Whitted. *An improved illumination model for shaded display.* Communication of the ACM, 23(6), 1980. Seiten: 343-349