Pixel Round

Die Mathematik des Projekts

Technische und didaktische Dokumentation der mathematischen Grundlagen des Generators für Kreise, Ellipsen, Kugeln und Ellipsoide in Pixel Art und Voxel Art. Zu jedem Konzept werden die formale Definition, die geometrische Begründung und die direkte Entsprechung im Quellcode des Projekts dargestellt.

Behandelte Gebiete: analytische Geometrie · diskrete Rasterung · digitale Topologie · Integralrechnung · lineare Algebra · Trigonometrie · Audiosynthese.

Inhaltsverzeichnis

1. Übersicht: mathematische Natur des Problems

Pixel Round ist ein Generator gerundeter Formen für **Pixel Art** und **Voxel Art**: Kreise und Ellipsen in 2D, Kugeln und Ellipsoide in 3D. Das zentrale Problem gehört zur Klasse der *diskreten Rasterung*: Gegeben eine analytisch über den reellen Zahlen $\mathbb{R}$ definierte Kurve oder Fläche ist zu bestimmen, welche Zellen eines regelmäßigen Gitters (Pixel in 2D, Voxel in 3D) zu markieren sind, um sie darzustellen.

Dieses Problem besitzt keine eindeutige Lösung. Verschiedene Einschlusskriterien erzeugen verschiedene diskrete Silhouetten, und jedes Kriterium hat seine eigene mathematische Begründung. Aus diesem Grund implementiert das Projekt drei verschiedene 2D-Algorithmen, drei Renderingmodi und drei 3D-Schattierungsstile, die in den folgenden Abschnitten beschrieben werden.

Die Ausführung erfolgt vollständig im Browser, ohne Serverkomponente, was an die mathematische Formulierung eine zusätzliche Anforderung an *Recheneffizienz* stellt. Die einbezogenen theoretischen Gebiete sind:

- analytische Geometrie der Kegelschnitte und Quadriken;
- inkrementelle Algorithmen mit Ganzzahlarithmetik (Bresenham);
- digitale Topologie (4-, 6- und 26-zusammenhängende Nachbarschaften);
- Integralrechnung und Zählmaß;
- lineare Algebra (Schnittebenen, perspektivische Projektion);
- Trigonometrie und Kugelkoordinaten;
- Signalsynthese und grundlegende Akustik.

2. Koordinatensystem und das diskrete Pixelgitter

Die Leinwand ist ein Gitter aus $G_x \times G_y$ Zellen. Jede Zelle (i, j) belegt das Quadrat $[i, i+1] \times [j, j+1]$. In stetiger Sicht liegt der **Mittelpunkt** der Zelle bei $(i + \frac{1}{2},\ j + \frac{1}{2})$. Diese Konvention ist entscheidend: Den Kreis mit der *Ecke* (i, j) oder mit dem *Mittelpunkt* $(i + \frac{1}{2}, j + \frac{1}{2})$ zu vergleichen ändert, welche Zellen als zur Form gehörig betrachtet werden.

$$\text{Mittelpunkt der Zelle } (i, j) \;=\; (i + \tfrac{1}{2},\ \ j + \tfrac{1}{2})$$

Für eine Form mit Durchmesser D erweitert der Code das Gitter auf $D+2$ Zellen pro Achse (eine Zelle Rand auf jeder Seite). Dies stellt sicher, dass äußere Pixel (N, S, O, W) nie auf den Matrixrand fallen. Der **Mittelpunkt** der Form liegt bei $(G_x/2,\ G_y/2)$, und die *Halbachsen* sind $r_x = D/2$ und $r_y = D/2$ (oder $W/2$ und $H/2$ bei einer Ellipse).

3. Die Grundgleichung: Kreis und Ellipse

Die gesamte Theorie des Projekts beginnt mit der impliziten Gleichung der im Ursprung zentrierten Ellipse mit Halbachsen r_x und r_y:

$$\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2 = 1$$

Wenn $r_x = r_y = r$, entartet die Ellipse zu einem **Kreis** vom Radius r. Punkte im *Inneren* erfüllen ≤ 1; Punkte im *Äußeren*, > 1. Dies ist die Ungleichung, die definiert, ob ein Pixel zur Form gehört oder nicht. Die drei Algorithmen sind im Grunde drei verschiedene Wege, diesen Test auf einem Gitter anzuwenden (oder zu approximieren).

Verschiebt man den Mittelpunkt nach (c_x, c_y) und wertet man im Zellmittelpunkt aus:

$$d_x = \frac{i + \frac{1}{2} - c_x}{r_x}, \qquad d_y = \frac{j + \frac{1}{2} - c_y}{r_y}, \qquad \text{innen} \iff d_x^2 + d_y^2 \leq 1$$

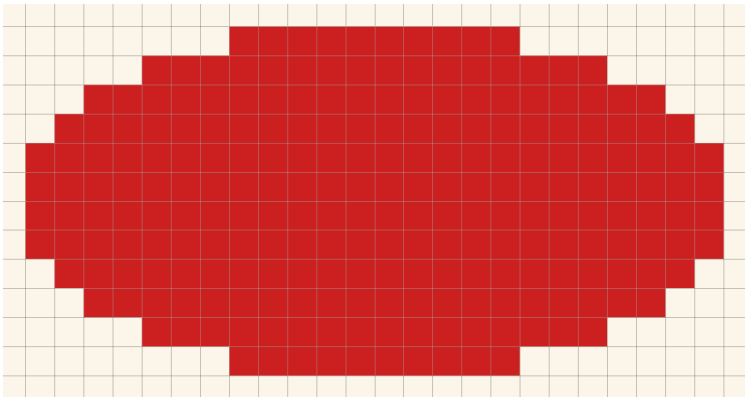

Abbildung 1: Ellipse mit $W = 24$ und $H = 12$ auf dem diskreten Raster.

4. Euklidischer Algorithmus (Abstandstest)

Idee. Für jede Zeile j analytisch die Spalten i finden, die innerhalb der Ellipse liegen. Löst man die Ellipsengleichung nach x bei gegebenem y:

$$x = c_x \pm r_x \sqrt{1 - \left(\frac{y - c_y}{r_y}\right)^2}$$

Für eine Zeile j mit vertikaler Verschiebung $d_y = j + \frac{1}{2} - c_y$ beträgt die *horizontale Halbbreite* der Ellipse in dieser Höhe:

$$dxM = r_x \sqrt{1 - \frac{d_y^2}{r_y^2}}$$

Falls $d_y^2 / r_y^2 \geq 1$, schneidet die Zeile die Ellipse nicht und wird übersprungen. Andernfalls werden alle Spalten i gefüllt, für die $c_x - dxM \leq i + \frac{1}{2} \leq c_x + dxM$ gilt. Die ganzzahligen Grenzen ergeben sich aus:

$$lo = \lceil c_x - dxM - \tfrac{1}{2} \rceil, \qquad hi = \lfloor c_x + dxM - \tfrac{1}{2} \rfloor$$

Begründung. Der Term $+\frac{1}{2}$ folgt aus der Konvention, dass die Zelle mit Index i ihren Mittelpunkt bei $i + \frac{1}{2}$ hat. Die Funktionen $\lceil \cdot \rceil$ und $\lfloor \cdot \rfloor$ wandeln das kontinuierliche Intervall in ganzzahlige Indizes um und stellen sicher, dass nur Zellen einbezogen werden, deren **Mittelpunkt** im Inneren der Ellipse liegt. Diese Formulierung liefert die diskrete Approximation, die der stetigen Kurve am nächsten kommt, und folglich die Silhouette mit der größten visuellen Glätte. Bei kleinen Durchmessern kann das Ergebnis jedoch im Vergleich zu den anderen Algorithmen einen weniger ausgeprägten Pixel-Art-Charakter aufweisen.

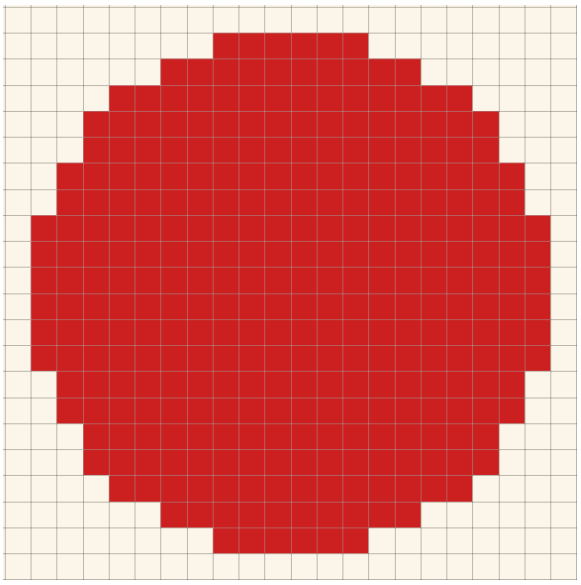

Abbildung 2: Euklidischer Algorithmus bei $D = 20$.

5. Bresenham-Algorithmus (ganzzahliger Mittelpunkt)

Der Bresenham-Algorithmus, 1965 für das Zeichnen von Liniensegmenten veröffentlicht, wurde später auf Kreise erweitert und von Pitteway und Kappel auf beliebige Ellipsen verallgemeinert. Sein Alleinstellungsmerkmal ist der Verzicht auf Gleitkommaoperationen und Wurzelziehen: Die Bestimmung des nächsten Pixels verwendet ausschließlich **Additionen und Vergleiche über Ganzzahlen**, vermittelt durch eine *Entscheidungsfunktion*, die auswertet, auf welcher Seite der analytischen Kurve der Mittelpunkt zwischen den beiden Kandidatenpixeln liegt.

5.1 Kreisfall

Für einen Kreis mit ganzzahligem Radius R beginnt man bei $(x{=}0,\, y{=}R)$ mit dem anfänglichen Entscheidungsparameter:

$$d_0 \;=\; 1 - R$$

In jedem Schritt (im Oktanten von 0 bis 45°):

$$\begin{cases} \text{wenn } d < 0: & \text{nächstes ist } (x{+}1,\; y), \quad d \mathrel{+}= 2x + 3 \\ \text{wenn } d \geq 0: & \text{nächstes ist } (x{+}1,\; y{-}1), \quad d \mathrel{+}= 2(x - y) + 5 \end{cases}$$

Begründung. Die Funktion d approximiert in jeder Iteration den Wert der impliziten Kreisgleichung am **Mittelpunkt** zwischen den beiden Pixel-Kandidaten:

$$F\big(x+1,\; y - \tfrac{1}{2}\big) \;=\; (x+1)^2 + \big(y - \tfrac{1}{2}\big)^2 - R^2$$

Das Vorzeichen von d gibt an, ob der Mittelpunkt im Inneren des Kreises liegt (in diesem Fall geht es nur horizontal weiter) oder im Äußeren (in diesem Fall wird auch die vertikale Koordinate verringert). Die acht Oktanten des Kreises ergeben sich durch Anwendung der Symmetrien der Diedergruppe D_4 auf den berechneten Oktanten, was im Code den acht `span(...)`-Aufrufen entspricht. Der erzeugte Strich zeigt das treppenartige Muster, das diskrete Approximationen glatter Kurven kennzeichnet.

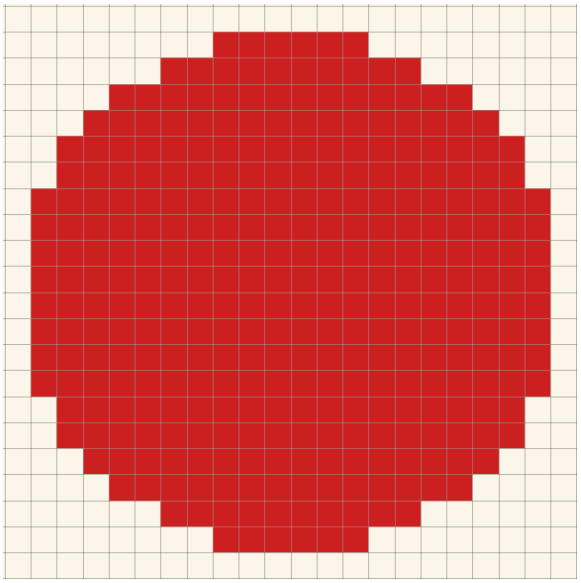

Abbildung 3: Bresenham-Algorithmus bei $D = 20$.

5.2 Elliptischer Fall (zwei Regionen)

Für eine Ellipse mit Halbachsen a, b teilt der Algorithmus den ersten Quadranten in **zwei Regionen**, getrennt durch den Punkt, an dem die Tangente einen Winkel von 45° bildet:

$$\text{Region I: } 2b^2x < 2a^2y \quad (|dy/dx| < 1), \qquad \text{Region II: sonst}$$

Die anfänglichen Entscheidungsparameter sind:

$$p_1 = b^2 - a^2b + \tfrac{a^2}{4},$$
$$p_2 = b^2\left(x + \tfrac{1}{2}\right)^2 + a^2(y-1)^2 - a^2b^2.$$

In jeder Iteration wird p durch vorab berechnete Inkremente aktualisiert (alle ganzzahlig nach Multiplikation mit 4). Das Ergebnis ist die minimal mögliche Arithmetik pro Pixel: *ein Vorzeichentest und zwei Additionen.* Der Code verwendet `Math.floor` bei der Berechnung von a und b aus r_x und r_y, um zu verhindern,

dass eine zusätzliche Zeile außerhalb der *Bounding Box* hinzugefügt wird, wenn r_y nicht ganzzahlig ist.

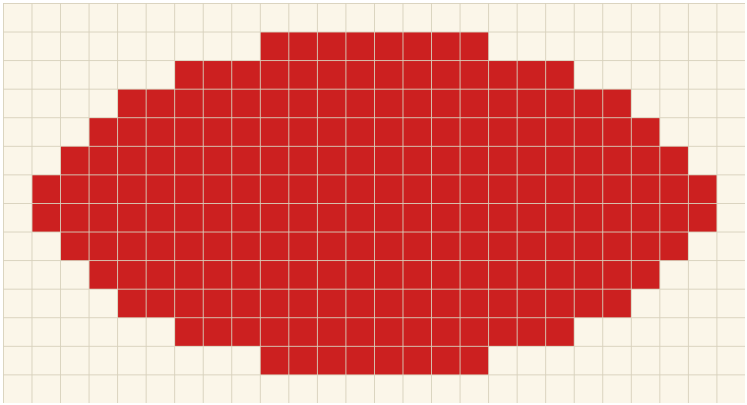

Abbildung 4: Bresenham-Algorithmus angewendet auf die Ellipse $W = 24$, $H = 12$.

6. Schwellwert-Algorithmus (Threshold / Eckenabdeckung)

Dieser Algorithmus fragt: *Liegt irgendeine Ecke der Zelle innerhalb der Ellipse?* Für jede Zelle (i, j) sucht der Code die dem Formmittelpunkt nächstgelegene Ecke (Projektion des Mittelpunkts auf die Zellkanten):

$$n_x = \text{clamp}(c_x,\, i,\, i{+}1), \quad n_y = \text{clamp}(c_y,\, j,\, j{+}1)$$

$$\text{innen} \iff \left(\frac{n_x - c_x}{r_x}\right)^2 + \left(\frac{n_y - c_y}{r_y}\right)^2 \leq 0{,}94$$

Der Wert 0,94 ist ein *empirischer Schwellwert* leicht unter 1,0. Seine Aufgabe ist es, das als *tangentialer Spike* von 1 Pixel bekannte Artefakt zu beseitigen, das an den vier Himmelsrichtungspunkten (N, S, O, W) auftritt, wo die Kurve eine ganze Kante der Zelle tangiert. Ohne diese Reduktion würde der Algorithmus eine zusätzliche Zelle einbeziehen, deren Beitrag die visuelle Darstellung der Form nicht verbessert.

Unter den drei Algorithmen erzeugt dieser die flächengrößte Silhouette für denselben Durchmesser D, da die Einschlussbedingung am wenigsten restriktiv ist: Es genügt, dass eine einzige Ecke der Zelle die Ellipsenungleichung erfüllt, damit die gesamte Zelle gefüllt wird.

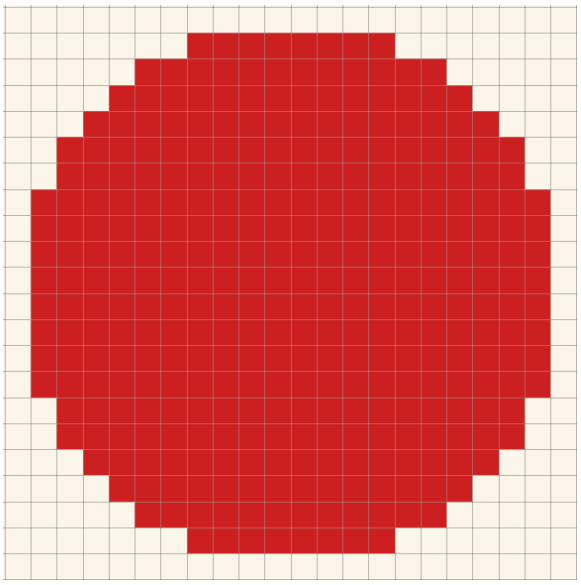

Abbildung 5: Schwellenwert-Algorithmus bei $D = 20$.

7. Renderingmodi: Gefüllt, Dünn, Dick

Mit einer Matrix $f[j][i] = 1$ für innere Zellen leitet das Projekt drei visuelle Stile durch **diskrete Topologie** ab:

7.1 Gefüllt

Gibt f unverändert zurück. Alle inneren Pixel.

7.2 Dünn (outline)

Ein Pixel gehört zur dünnen Kontur, wenn es gefüllt ist *und* mindestens einen orthogonalen Nachbarn (N, S, O, W) *außerhalb* der Form hat. In logischer Notation:

$$b(i,j) \;=\; f(i,j) \;\wedge\; (\neg f(i{-}1,j) \vee \neg f(i{+}1,j) \vee \neg f(i,j{-}1) \vee \neg f(i,j{+}1))$$

Geometrisch: Es ist der **diskrete Rand** der Form, das diskrete Analogon des topologischen Randes $\partial F = F \cap \overline{F^{\mathrm{c}}}$.

7.3 Dick (thick)

Der Dick-Modus fügt zu den Randpixeln die **Diagonalbrücken** hinzu: innere Zellen, die gleichzeitig einen horizontalen *und* einen vertikalen Randnachbarn besit-

zen. Dies schließt die 1-Pixel-Diagonalen, die der Dünn-Modus offen lässt, nützlich wenn die Kontur in einer Szene stabil wirken muss (keine 45°-Löcher).

$$\begin{aligned} t(i,j) &= b(i,j) \;\vee\; \big(f(i,j) \wedge h_H \wedge h_V\big), \\ h_H &= b(i{-}1,j) \;\vee\; b(i{+}1,j), \\ h_V &= b(i,j{-}1) \;\vee\; b(i,j{+}1). \end{aligned}$$

8. Berechnung der diskreten Fläche

Die Funktion `area2D` **zählt** einfach die in f markierten Zellen. Dies ist das *Zählmaß*, das das Riemann-Integral der Indikatorfunktion der Ellipse approximiert:

$$\underbrace{\pi\, r_x\, r_y}_{\text{stetige Fläche}} \quad\longleftrightarrow\quad \underbrace{\sum_{j=0}^{G_y-1} \sum_{i=0}^{G_x-1} f(i,j)}_{\text{diskrete Fläche}}$$

Für großes D gilt diskrete Fläche/stetige Fläche $\to 1$. Für kleines D ist der Unterschied zwischen den Algorithmen sichtbar: Threshold neigt zur Überschätzung; Euklidisch bleibt nahe an der idealen Fläche; Bresenham liegt irgendwo dazwischen.

9. Von 2D zu 3D: die Gleichung des Ellipsoids

In 3D ist die Grundform das **Ellipsoid**, zentriert bei (c_x, c_y, c_z) mit Halbachsen r_x, r_y, r_z. Seine Gleichung ist die natürliche Verallgemeinerung der Ellipse:

$$\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 \;=\; 1$$

Wenn $r_x = r_y = r_z$, erhalten wir die **Kugel** zurück. Auswertung im Zentrum jedes Voxels:

$$a_\xi = \frac{\xi + \frac{1}{2} - c_\xi}{r_\xi} \;\; (\xi \in \{x,y,z\}), \qquad \text{innen} \iff a_x^2 + a_y^2 + a_z^2 \le 1$$

10. Voxelisierung und Volumenberechnung

Das stetige Volumen des Ellipsoids ist ein klassisches Ergebnis der Integralrechnung, gewonnen durch Integration über Scheiben:

$$V = \frac{4}{3}\pi\, r_x\, r_y\, r_z$$

Die diskrete Version durchläuft jedes Voxel der Box $D_x \times D_y \times D_z$ und erhöht einen Zähler, wenn $a_x^2 + a_y^2 + a_z^2 \leq 1$. Dies ist der Algorithmus `voxelVolume`.

Schale (shell) vs. Inneres. Für den Modus *filled* zeigt das Projekt nur Voxel mit mindestens einem 6-Nachbarn außerhalb der erhaltenen Menge (also Voxel, die von der Außenfläche oder von der Schnittfläche aus sichtbar sind). Für *thin* wird nur die **Oberfläche** des vollständigen Ellipsoids gezeigt (1 Voxel Dicke).

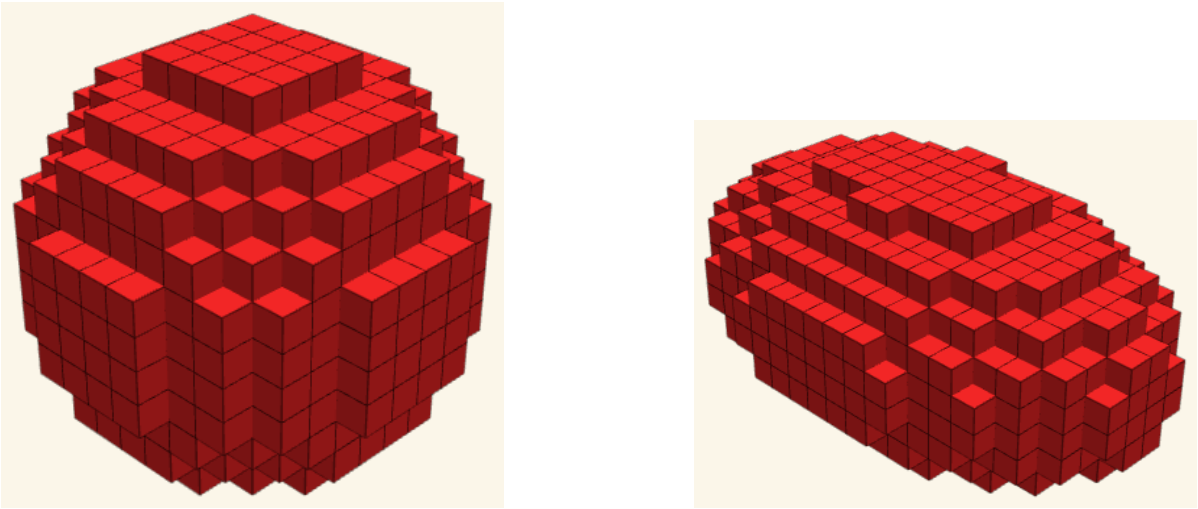

Abbildung 6: Voxelisierte Kugel ($D = 12$) und voxelisiertes Ellipsoid ($W = 20$, $H = 10$, $D = 12$).

11. Geometrische Schnitte: Ebenen und der 45°-Diagonalschn

Pixel Round erlaubt es, den Körper durch **drei Ebenen** zu schneiden. Jeder Schnitt ist eine *lineare Ungleichung*, die Voxel verwirft:

$$\begin{array}{lcll} \text{X-Achse:} & x \geq \mathit{cut} & \Rightarrow & \text{verworfen} \\ \text{Y-Achse:} & y \geq \mathit{cut} & \Rightarrow & \text{verworfen} \\ \text{Diagonale:} & x + y \geq \mathit{cut} & \Rightarrow & \text{verworfen} \end{array}$$

Die X- und Y-Ebenen sind senkrecht zu den Koordinatenachsen, mit Normalenvektoren $(1, 0, 0)$ und $(0, 1, 0)$. Die *Diagonalebene* hat Normalenvektor $(1, 1, 0)/\sqrt{2}$

und schneidet bei 45° in der XY-Ebene: mathematisch ist es die Schar der Ebenen $x + y = k$. Da das Maximum von $x + y$ in einer Box $D_x \times D_y$ gleich $D_x + D_y$ ist, hat der Schieberegler für den Diagonalschnitt den Bereich $D_x + D_y$, während X und Y die Bereiche D_x bzw.
D_y haben. Der Schnitt ist **proportional**: Der Code speichert einen Prozentsatz *cutPct* und berechnet die Grenze als

$$cut \;=\; cutPct \cdot \max_{\text{Achse}}$$

so dass 50% auf einer Achse 50% bleibt, wenn der Benutzer die Achse wechselt oder die Größe ändert.

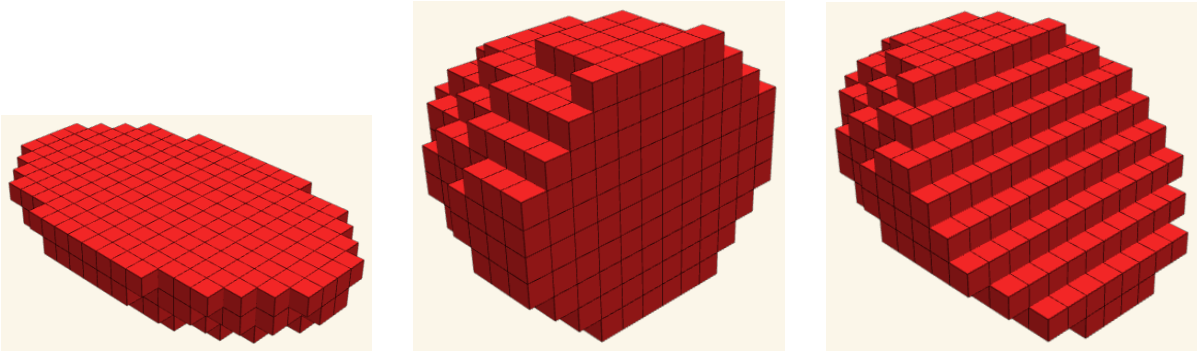

Abbildung 7: Ebene Schnitte bei 50% entlang Y-, X- und Diagonalachse (links nach rechts).

12. Dicker 3D-Modus: Dreidimensionalisierung durch Schichten

Für den *thick*-Modus in 3D wendet das Projekt den 2D-Dick-Algorithmus auf **alle Schichten** entlang der drei Achsen an: XY für jedes z, XZ für jedes y, YZ für jedes x. Anschließend wird die *Vereinigung* der Ergebnisse gebildet. Die geometrische Motivation ist die Dichtigkeit: die minimale Menge von Voxeln, die alle Diagonalen schließt, ohne die Schalendicke zu verdoppeln. Formal, mit $S_z(z)$ als XY-Schicht in Höhe z und T als 2D-Dick-Operator:

$$thick3D \;=\; \bigcup_{z} T(S_z(z)) \;\cup\; \bigcup_{y} T(S_y(y)) \;\cup\; \bigcup_{x} T(S_x(x))$$

Das Ergebnis wird dann mit der vom Schnitt erhaltenen Menge geschnitten. Die zugrundeliegende Theorie ist die **digitale Topologie**: Der Dick-Operator garantiert die *26-Konnektivität* der Schale (Nachbarn einschließlich Diagonalen), nützlich für den Bau in Minecraft, wo diagonal benachbarte Voxel sich nicht über Flächen berühren.

13. 3D-Kamera: Kugelkoordinaten

Die 3D-Kamera umkreist das Objekt in Abstand r mit zwei Winkeln — θ (Azimut, in der horizontalen Ebene) und φ (Polarwinkel, von der vertikalen Achse aus). Dies ist die **physikalische Konvention** der Kugelkoordinaten, und die Umrechnung in kartesische (die Form, die *three.js* versteht) lautet:

$$\begin{aligned} x &= r\,\sin(\varphi)\,\cos(\theta), \\ y &= r\,\cos(\varphi), \\ z &= r\,\sin(\varphi)\,\sin(\theta). \end{aligned}$$

Anfangsposition: $\theta = \pi/4$ (45°, isometrische Perspektive) und $\varphi = \pi/3$ (60° vom Nordpol, leicht über dem Äquator). Mausziehen erhöht θ und φ direkt; Mausrad/Pinch erhöht r. Die Einfachheit dieser Parametrisierung erlaubt freie Rotation ohne Quaternionen oder explizite Matrizen.

14. Auto-Zoom: umschließende Kugel und FOV

Wenn der Benutzer die Größe der Figur ändert oder die Kamera zurücksetzt, berechnet das Projekt den **idealen Abstand** der Kamera neu, sodass das Objekt aus jedem Winkel in den Bildausschnitt passt. Die Idee ist, das Objekt in eine *Kugel mit Radius* R einzuschließen (die Hälfte der Diagonalen der AABB, axis-aligned bounding box):

$$R \;=\; \sqrt{(D_x/2)^2 + (D_y/2)^2 + (D_z/2)^2}$$

Eine Kugel hat eine **unter Rotation invariante Projektion**, also wenn sie in einer Orientierung in den Bildausschnitt passt, passt sie in allen. Bei vertikalem FOV der Kamera ($fov_V = 35°$ im Bogenmaß) ist der nötige Abstand, um eine Kugel mit Radius R einzurahmen:

$$dist_V \;=\; \frac{R}{\tan(fov_V/2)}$$

Das horizontale FOV ergibt sich aus dem Seitenverhältnis der Leinwand ($aspect =$ Breite/Höhe) durch:

$$fov_H \;=\; 2\,\arctan\big(\,\tan(fov_V/2)\cdot aspect\,\big)$$

Der Endabstand ist das Maximum von $dist_V$ und $dist_H$, multipliziert mit 1,20 (20% Sichtrand). Wenn die Figur geschnitten ist, verkleinert der Code D_x oder D_y auf die verbleibende Größe, damit eine zu 75% geschnittene Kugel nicht winzig in einer Ecke erscheint.

15. Schattierung (Shading) und RGB-Farbarithmetik

Die drei 3D-Stile (*Classic*, *Smooth*, *Blocks*) unterscheiden sich durch **multiplikative Faktoren**, die auf die drei sichtbaren Flächen jedes Voxels angewendet werden (Oberseite, beleuchtete Seite, dunkle Seite). Die Funktion `shadeHex(hex, factor)` parst das Hex in (R, G, B) und multipliziert jeden Kanal, mit *Clamp* auf $[0, 255]$:

$$R' = \text{clamp}(R \cdot f), \quad G' = \text{clamp}(G \cdot f), \quad B' = \text{clamp}(B \cdot f)$$

Dies ist eine *lineare Approximation* der idealen Lambertschen Beleuchtungsfunktion,

$$I \;=\; \max(0,\; \mathbf{n} \cdot \mathbf{l}) \cdot \textit{Grundfarbe},$$

wenn die drei Faktoren dem Kosinus des Winkels zwischen der Normalen jeder Fläche und einer festen Lichtrichtung entsprechen. In *Smooth* sind die Faktoren nahe beieinander (ähnliche Flächen); in *Classic* unterscheiden sich die Faktoren stärker (starker Kontrast). *Blocks* führt zusätzlich eine geometrische Vertiefung in den Voxeln ein — eine konstante Verschiebung jeder Fläche nach innen.

16. Audio: Frequenzen und die musikalische Skala

Die Klänge der Oberfläche werden in Echtzeit über WebAudio synthetisiert. Jeder Klang ist eine **reine Sinuswelle**:

$$s(t) \;=\; A \cdot \sin(2\pi\, f\, t)$$

Die vom Projekt gewählten Werte von f (in Hz) sind nicht willkürlich — sie approximieren musikalische Noten der *gleichstufigen 12-Ton-Stimmung*, in der jeder Halbton die Frequenz mit $2^{1/12} \approx 1{,}0595$ multipliziert:

Klang	Frequenz (Hz)	Nächste Note	Verwendung
click	620	≈ D#5 / E♭5	Schaltflächen
hover	1100	≈ C#6	Desktop-Hover
tick	1300	≈ E6	Schritte des Schiebereglers
ok	520 → 720 → 920	aufsteigendes Arpeggio	Reset / Download
pop	680 + 980	Intervall der reinen Quinte	Umschalter
error	200	≈ G#3	ungültige Aktion

Reine Quinte (Verhältnis 3 : 2): Der *pop*-Klang mischt 680 Hz und 980 Hz; $980/680 \approx 1{,}44$ (eine reine Quinte wäre 1,5), genug, um konsonant zu klingen.

Das **Arpeggio des *ok*-Klangs** weist aufeinanderfolgende Verhältnisse nahe an Terzen auf: 520 → 720 (Verhältnis 1,38, nahe der temperierten kleinen Terz von 1,2 leicht erweitert) und 720 → 920, eine Konfiguration, die eine als konklusiv wahrgenommene aufsteigende Kadenz erzeugt. Die Hüllkurve hat eine Dauer zwischen 12 und 160 ms und einen Spitzenpegel von 0,035 (etwa −29 dBFS), eine bewusst niedrige Lautstärke, um auditive Ermüdung bei längerer Nutzung zu vermeiden.

17. Schlussfolgerung

Das vorliegende Dokument beschrieb systematisch die mathematischen Grundlagen, die in Pixel Round verwendet werden. In etwa tausend Zeilen JavaScript-Code integriert das Projekt Beiträge aus mehreren Gebieten:

- **Analytische Geometrie**: implizite Gleichungen der Ellipse und des Ellipsoids als primäres Zugehörigkeitskriterium.
- **Klassische Rasterungsalgorithmen**: Bresenham in der Mittelpunktsformulierung, mit der Pitteway/Kappel-Erweiterung für Ellipsen.
- **Digitale Topologie**: diskrete Randoperatoren, Konzepte der 4-, 6- und 26-Konnektivität und schichtbasierte Komposition.
- **Integralrechnung**: Volumen des Ellipsoids als Dreifachintegral und sein diskretes Gegenstück (Zählmaß über Voxel).
- **Lineare Algebra**: durch lineare Ungleichungen definierte Schnittebenen und dreidimensionale perspektivische Projektion.
- **Trigonometrie**: Parametrisierung der Kamera in Kugelkoordinaten und automatische Anpassung der Distanz in Abhängigkeit vom FOV.
- **Signalsynthese**: reine Sinuswellen und Approximation der gleichstufigen Zwölftonstimmung.

Die zentrale Beobachtung, die die Studie zu formulieren erlaubt, lautet: Auf einem diskreten Gitter **gibt es keine einzige korrekte Darstellung** einer stetigen Kurve. Jeder in diesem Dokument vorgestellte Algorithmus entspricht einer eigenen mathematischen Definition des Zellaufnahmekriteriums, und jede Definition erzeugt eine Silhouette mit eigenen formalen Eigenschaften — Glätte beim Euklidischen Algorithmus, Treppenmuster bei Bresenham, größere Abdeckung beim Eckenabdeckungskriterium. Die Wahl des Algorithmus ist daher eine technische

Entscheidung, die das beabsichtigte Darstellungsziel und die gewünschten metrischen Eigenschaften der resultierenden Form berücksichtigen muss.