Pixel Round

La Matemática del Proyecto

Documentación técnica y didáctica de los fundamentos matemáticos empleados en el generador de círculos, elipses, esferas y elipsoides para pixel art y voxel art. Para cada concepto se presentan la definición formal, la justificación geométrica y la correspondencia directa con el código fuente del proyecto.

Áreas tratadas: geometría analítica · rasterización discreta · topología digital · cálculo integral · álgebra lineal · trigonometría · síntesis de audio.

Índice

1. Visión general: naturaleza matemática del problema

Pixel Round es un generador de formas redondeadas para **pixel art** y **voxel art**: círculos y elipses en 2D, esferas y elipsoides en 3D. El problema central pertenece a la clase de la *rasterización discreta*: dada una curva o superficie definida analíticamente sobre los números reales $\mathbb{R}$, determinar qué celdas de una rejilla regular (píxeles en 2D, vóxeles en 3D) deben marcarse para representarla.

Este problema no admite solución única. Distintos criterios de inclusión producen distintas siluetas discretas, y cada criterio posee su propia justificación matemática. Por esta razón el proyecto implementa tres algoritmos 2D distintos, tres modos de renderizado y tres estilos de sombreado 3D, descritos en las secciones siguientes.

La ejecución ocurre íntegramente en el navegador, sin componente servidor, lo que impone a la formulación matemática un requisito adicional de *eficiencia computacional*. Las áreas teóricas involucradas son:

- geometría analítica de cónicas y cuádricas;
- algoritmos incrementales con aritmética entera (Bresenham);
- topología digital (vecindades 4-, 6- y 26-conectadas);
- cálculo integral y medida de conteo;
- álgebra lineal (planos de corte, proyección perspectiva);
- trigonometría y coordenadas esféricas;
- síntesis de señales y acústica fundamental.

2. Sistema de coordenadas y la rejilla discreta de píxeles

El lienzo es una rejilla de $G_x \times G_y$ celdas. Cada celda (i, j) ocupa el cuadrado $[i, i+1] \times [j, j+1]$. En código continuo, el **centro** de la celda se ubica en $(i + \frac{1}{2},\ j + \frac{1}{2})$. Esta convención es crítica: comparar la circunferencia con la *esquina* (i, j) o con el *centro* $(i + \frac{1}{2}, j + \frac{1}{2})$ cambia qué celdas se consideran pertenecientes a la forma.

$$\text{centro de la celda } (i,j) \;=\; \left(i+\tfrac{1}{2},\;\; j+\tfrac{1}{2}\right)$$

Para una forma de diámetro D, el código expande la rejilla a $D+2$ celdas por eje (margen de una celda a cada lado). Esto garantiza que los píxeles extremos (N, S, E, O) nunca caigan en el borde de la matriz. El **centro** de la forma queda en $(G_x/2,\ G_y/2)$, y los *semiejes* son $r_x = D/2$ y $r_y = D/2$ (o $W/2$ y $H/2$ para una elipse).

3. La ecuación fundamental: círculo y elipse

Toda la teoría del proyecto comienza con la ecuación implícita de la elipse centrada en el origen con semiejes r_x y r_y:

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

Cuando $r_x = r_y = r$, la elipse degenera en una **circunferencia** de radio r. Los puntos del *interior* satisfacen ≤ 1; los del *exterior*, > 1. Ésta es la desigualdad que define si un píxel pertenece o no a la forma. Los tres algoritmos son, en el fondo, tres maneras distintas de aplicar (o aproximar) este test en una rejilla.

Trasladando el centro a (c_x, c_y) y evaluando en el centro de cada celda:

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

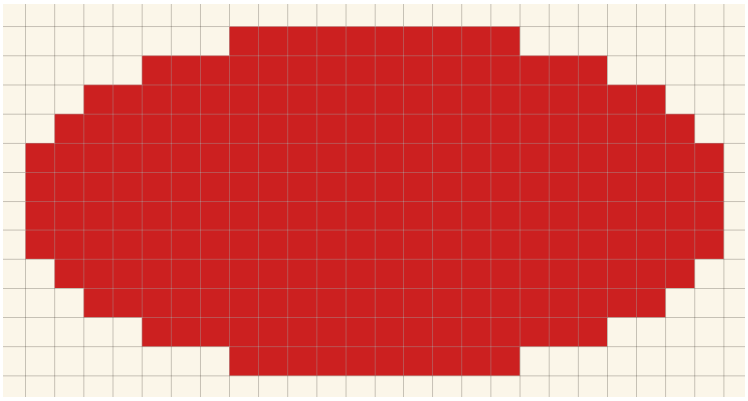

Figura 1: Elipse con $W = 24$ y $H = 12$ sobre la rejilla discreta.

4. Algoritmo Euclídeo (test de distancia)

Idea. Para cada fila j, hallar analíticamente las columnas i que están dentro de la elipse. Resolviendo la ecuación de la elipse para x, dado y:

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

Para una fila j cuyo desplazamiento vertical es $d_y = j + \frac{1}{2} - c_y$, la *semianchura horizontal* de la elipse a esa altura es:

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

Si $d_y^2 / r_y^2 \geq 1$, la fila no intersecta la elipse y se omite. En caso contrario, se rellenan todas las columnas i tales que $c_x - dxM \leq i + \frac{1}{2} \leq c_x + dxM$. Los límites enteros se obtienen de:

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

Justificación. El término $+\frac{1}{2}$ proviene de la convención de que la celda de índice i tiene su centro en $i + \frac{1}{2}$. Las funciones $\lceil \cdot \rceil$ y $\lfloor \cdot \rfloor$ realizan la conversión del intervalo continuo a índices enteros, garantizando que solo se incluyan las celdas cuyo **centro** pertenezca al interior de la elipse. Esta formulación produce la aproximación discreta más cercana a la curva continua y, en consecuencia, la silueta de mayor suavidad visual. Para diámetros pequeños, sin embargo, el resultado puede presentar un carácter de pixel art menos marcado en comparación con los demás algoritmos.

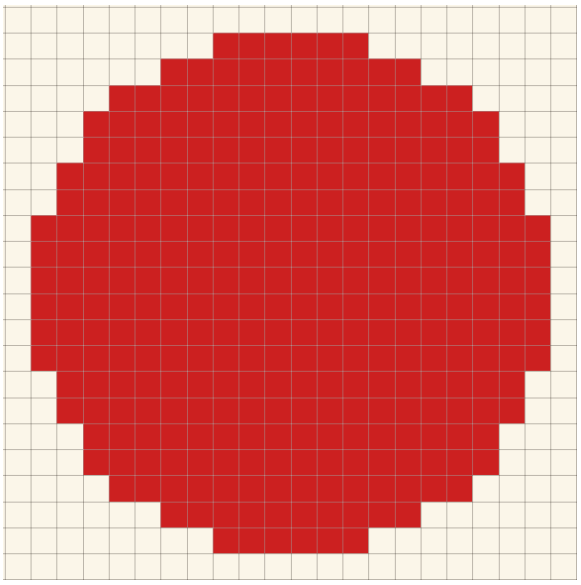

Figura 2: Algoritmo Euclídeo para $D = 20$.

5. Algoritmo de Bresenham (punto medio entero)

El algoritmo de Bresenham, publicado en 1965 para el trazado de segmentos de recta, fue posteriormente extendido a circunferencias y generalizado por Pitteway y Kappel a elipses arbitrarias. Su propiedad distintiva es prescindir de operaciones en punto flotante y extracción de raíz cuadrada: la determinación del píxel siguiente emplea exclusivamente **sumas y comparaciones sobre enteros**, mediante una *función de decisión* que evalúa de qué lado de la curva analítica se encuentra el punto medio entre los dos píxeles candidatos.

5.1 Caso circular

Para una circunferencia de radio entero R, se parte de $(x{=}0,\ y{=}R)$ con parámetro de decisión inicial:

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

En cada paso (en el octante de 0 a 45°):

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

Justificación. La función d aproxima, en cada iteración, el valor de la ecuación implícita de la circunferencia evaluado en el **punto medio** entre las dos opciones

de píxel candidato:

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

El signo de d indica si el punto medio se encuentra en el interior de la circunferencia (en cuyo caso se avanza solo horizontalmente) o en el exterior (en cuyo caso también se decrementa la coordenada vertical). Los ocho octantes de la circunferencia se obtienen aplicando las simetrías del grupo diedral D_4 al octante calculado, lo que corresponde, en el código, a las ocho llamadas a `span(...)`. El trazado resultante exhibe el patrón escalonado característico de las aproximaciones discretas de curvas suaves.

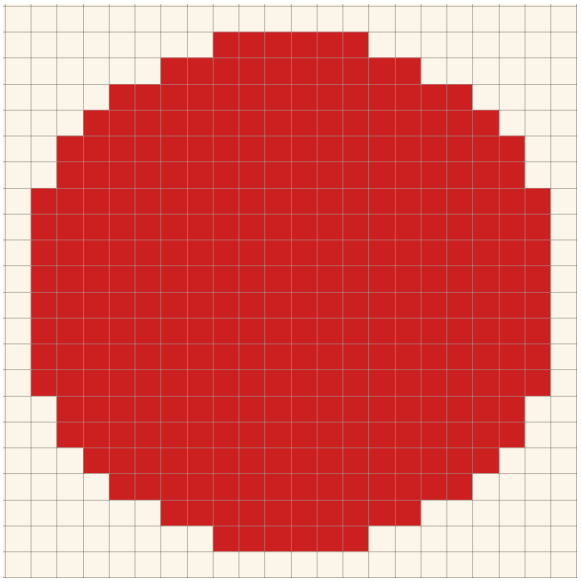

Figura 3: Algoritmo de Bresenham para $D = 20$.

5.2 Caso elíptico (dos regiones)

Para una elipse con semiejes a, b, el algoritmo divide el primer cuadrante en **dos regiones**, delimitadas por el punto donde la tangente forma 45°:

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

Los parámetros de decisión iniciales son:

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

En cada iteración, p se actualiza por incrementos precalculados (todos enteros, tras multiplicar por 4). El resultado es la aritmética mínima posible por píxel: *un*

test de signo y dos sumas. El código usa `Math.floor` al calcular a y b a partir de r_x y r_y para evitar añadir una fila extra fuera del *bounding box* cuando r_y no es entero.

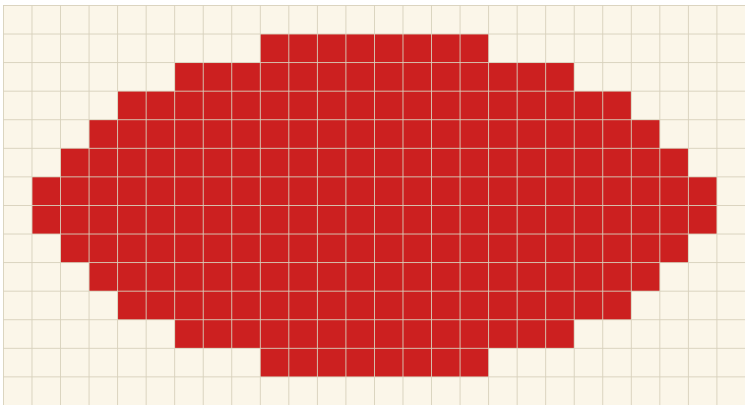

Figura 4: Algoritmo de Bresenham aplicado a la elipse $W = 24$, $H = 12$.

6. Algoritmo de Umbral (Threshold / cobertura de esquina)

Este algoritmo pregunta: *¿hay alguna esquina de la celda dentro de la elipse?* Para cada celda (i, j), el código busca la esquina más próxima al centro de la forma (proyección del centro sobre las aristas de la celda):

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

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

El valor 0,94 es un *umbral empírico* ligeramente inferior a 1,0. Su función es eliminar el artefacto conocido como *spike tangencial* de 1 píxel que aparece en los cuatro puntos cardinales (N, S, E, O), donde la curva es tangente a una arista completa de la celda. Sin esta reducción, el algoritmo incluiría una celda adicional cuya contribución no mejora la representación visual de la forma.

Entre los tres algoritmos, éste es el que produce la silueta de mayor área para un mismo diámetro D, ya que la condición de inclusión es la menos restrictiva: basta con que una sola esquina de la celda satisfaga la desigualdad de la elipse para que se rellene la celda entera.

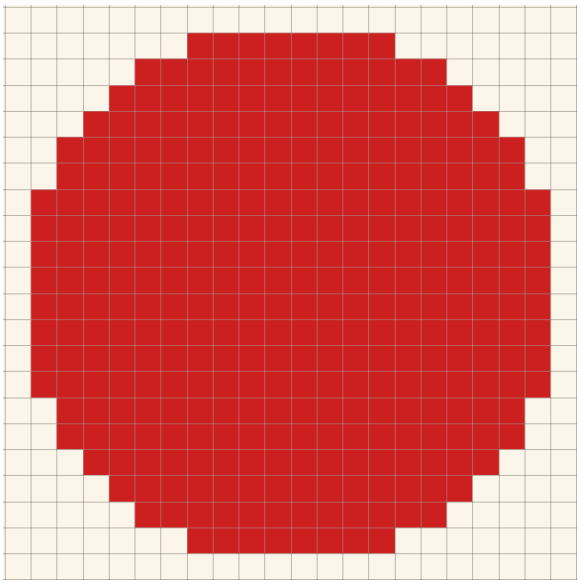

Figura 5: Algoritmo de Umbral para $D = 20$.

7. Modos de renderizado: Relleno, Fino, Grueso

Teniendo una matriz $f[j][i] = 1$ para celdas internas, el proyecto deriva tres estilos visuales por **topología discreta**:

7.1 Relleno

Devuelve f sin modificación. Todos los píxeles internos.

7.2 Fino (outline)

Un píxel pertenece al contorno fino si está relleno y tiene al menos un vecino ortogonal (N, S, E, O) *fuera* de la forma. En notación lógica:

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

Geométricamente: es el **borde discreto** de la forma, análogo discreto de la frontera topológica $\partial F = F \cap \overline{F^{\mathrm{c}}}$.

7.3 Grueso (thick)

El modo grueso añade a los píxeles del borde los **puentes diagonales**: celdas internas que tienen simultáneamente un vecino-borde horizontal y un vecino-borde vertical. Esto cierra las diagonales de 1 píxel que el modo fino deja abiertas, útil

cuando el contorno debe verse estable en una escena (sin agujeros a 45°).

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

8. Cálculo de área discreta

La función `area2D` simplemente **cuenta** las celdas marcadas en f. Esto es la *medida de conteo*, que aproxima la integral de Riemann de la función indicadora de la elipse:

$$\underbrace{\pi\, r_x\, r_y}_{\text{área continua}} \qquad \longleftrightarrow \qquad \underbrace{\sum_{j=0}^{G_y-1} \sum_{i=0}^{G_x-1} f(i,j)}_{\text{área discreta}}$$

Para D grande, área discreta/área continua $\to 1$. Para D pequeño, la diferencia entre algoritmos es visible: Threshold tiende a sobreestimar; Euclídeo se mantiene cerca del área ideal; Bresenham queda en algún lugar intermedio.

9. De 2D a 3D: la ecuación del elipsoide

En 3D, la forma fundamental es el **elipsoide** centrado en (c_x, c_y, c_z) con semiejes r_x, r_y, r_z. Su ecuación es la generalización natural de la elipse:

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

Cuando $r_x = r_y = r_z$, recuperamos la **esfera**. Evaluando en el centro de cada vóxel:

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

10. Voxelización y cálculo de volumen

El volumen continuo del elipsoide es un resultado clásico del cálculo integral, obtenido por integración sobre discos:

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

La versión discreta recorre cada vóxel de la caja $D_x \times D_y \times D_z$ e incrementa un contador cuando $a_x^2 + a_y^2 + a_z^2 \leq 1$. Éste es el algoritmo de `voxelVolume`.

Cáscara (shell) vs. interior. Para el modo *filled* el proyecto muestra solo los vóxeles con al menos un 6-vecino fuera del conjunto preservado (es decir, vóxeles visibles desde la superficie externa o desde la cara de corte). Para *thin*, solo se muestra la **superficie** del elipsoide completo (1 vóxel de espesor).

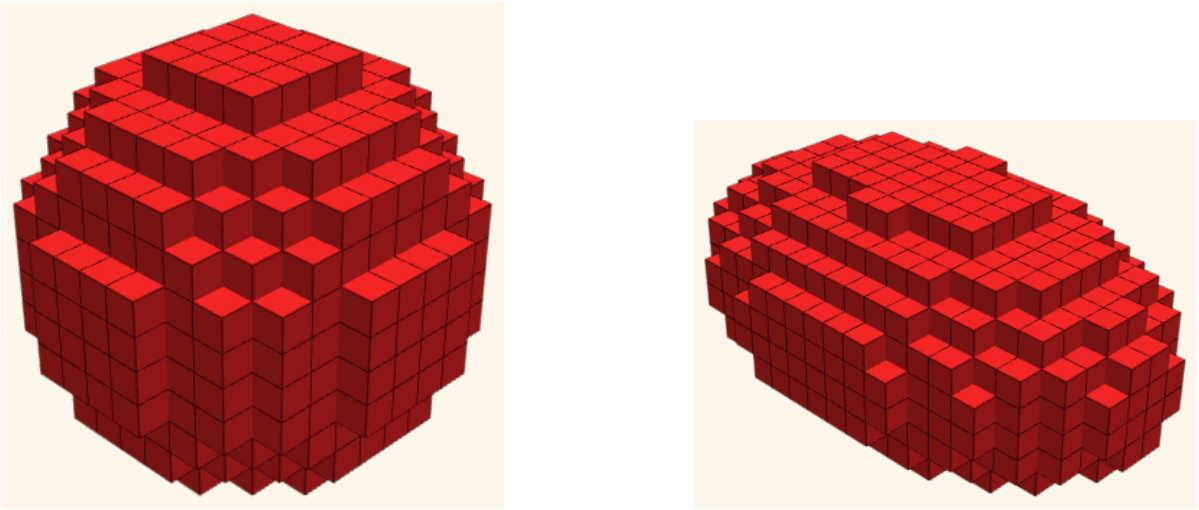

Figura 6: Esfera ($D = 12$) y elipsoide ($W = 20$, $H = 10$, $D = 12$) voxelizados.

11. Cortes geométricos: planos y el corte diagonal a 45°

Pixel Round permite cortar el sólido por **tres planos**. Cada corte es una *desigualdad lineal* que descarta vóxeles:

$$\begin{array}{lll} \text{Eje X:} & x \geq cut & \Rightarrow \text{ descartado} \\ \text{Eje Y:} & y \geq cut & \Rightarrow \text{ descartado} \\ \text{Diagonal:} & x + y \geq cut & \Rightarrow \text{ descartado} \end{array}$$

Los planos X e Y son perpendiculares a los ejes coordenados, con vectores normales $(1, 0, 0)$ y $(0, 1, 0)$. El *plano diagonal* tiene vector normal $(1, 1, 0)/\sqrt{2}$ y corta a 45° en el plano XY: matemáticamente es la familia de planos $x + y = k$. Como el máximo de $x + y$ en una caja $D_x \times D_y$ vale $D_x + D_y$, el deslizador del corte diagonal tiene amplitud $D_x + D_y$, mientras que X e Y tienen amplitudes D_x y D_y, respectivamente. El corte es **proporcional**: el código guarda un porcentaje $cutPct$ y recalcula el límite como

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

de modo que el 50% en un eje sigue siendo 50% cuando el usuario cambia de eje o redimensiona.

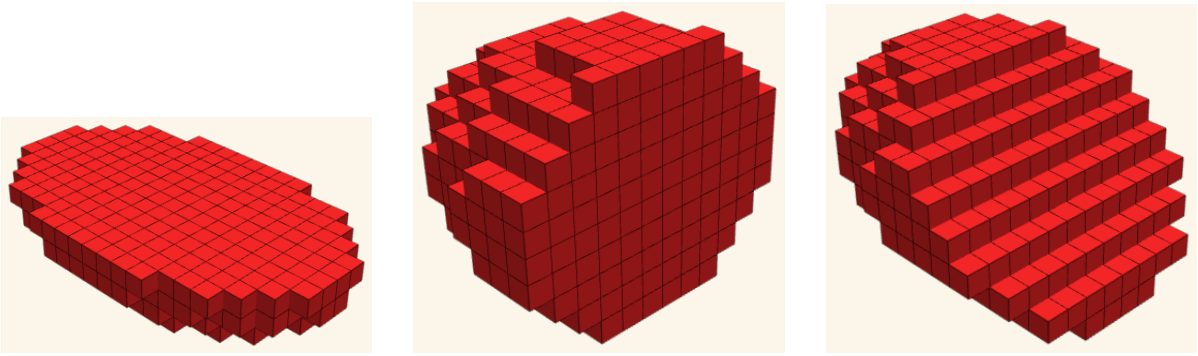

Figura 7: Cortes planos al 50% en los ejes Y, X y diagonal (de izquierda a derecha).

12. Modo Grueso 3D: tridimensionalización por rebanadas

Para el modo *thick* en 3D, el proyecto aplica el algoritmo grueso 2D a **todas las rebanadas** en los tres ejes: XY para cada z, XZ para cada y, YZ para cada x. A continuación hace la *unión* de los resultados. La motivación geométrica es la estanqueidad: el conjunto mínimo de vóxeles que tapan todas las diagonales sin duplicar el grosor de la cáscara. Formalmente, siendo $S_z(z)$ la rebanada XY a altura z y T el operador grueso 2D:

$$\mathit{thick3D} = \bigcup_z T(S_z(z)) \cup \bigcup_y T(S_y(y)) \cup \bigcup_x T(S_x(x))$$

El resultado se intersecta luego con el conjunto preservado por el corte. La teoría subyacente es la **topología digital**: el operador grueso garantiza la *26-conectividad* de la cáscara (vecinos incluyendo diagonales), útil para construir en Minecraft donde los vóxeles diagonales no se tocan por cara.

13. Cámara 3D: coordenadas esféricas

La cámara 3D orbita el objeto a distancia r con dos ángulos — θ (acimut, en el plano horizontal) y φ (polar, desde el eje vertical). Ésta es la **convención física** de coordenadas esféricas, y la conversión a cartesianas (la forma que entiende *three.js*) es:

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

Posición inicial: $\theta = \pi/4$ (45°, perspectiva isométrica) y $\varphi = \pi/3$ (60° desde el polo Norte, ligeramente por encima del ecuador). Arrastrar el ratón incrementa θ y φ directamente; rueda/pellizco incrementa r. La simplicidad de esta parametrización es lo que permite la rotación libre sin cuaterniones ni matrices explícitas.

14. Auto-zoom: esfera envolvente y FOV

Cuando el usuario cambia el tamaño de la figura o reinicia la cámara, el proyecto recalcula la **distancia ideal** de la cámara para que el objeto encaje en la pantalla en cualquier ángulo. La idea es encerrar el objeto en una *esfera de radio* R (la mitad de la diagonal del AABB, axis-aligned bounding box):

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

Una esfera tiene **proyección invariante por rotación**, así que si encaja en la pantalla en una orientación, encaja en todas. Dado el FOV vertical de la cámara ($fov_V = 35°$ en radianes), la distancia necesaria para encuadrar una esfera de radio R es:

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

El FOV horizontal sale de la razón de aspecto del lienzo ($aspect$ = ancho/alto) por:

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

La distancia final es el máximo entre $dist_V$ y $dist_H$, multiplicado por 1,20 (20% de margen visual). Cuando la figura está cortada, el código encoge D_x o D_y al tamaño restante, de modo que una esfera cortada al 75% no aparezca minúscula en una esquina.

15. Tonos (shading) y aritmética de color RGB

Los tres estilos 3D (*Classic*, *Smooth*, *Blocks*) se diferencian por **factores multiplicativos** aplicados a las tres caras visibles de cada vóxel (techo, lado iluminado, lado oscuro). La función `shadeHex(hex, factor)` parsea el hex en (R, G, B) y multiplica cada canal, con *clamp* en $[0, 255]$:

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

Esto es una *aproximación lineal* de la función de iluminación Lambertiana ideal,

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

cuando los tres factores corresponden al coseno del ángulo entre la normal de cada cara y una dirección de luz fija. En *Smooth* los factores son próximos (caras parecidas); en *Classic* los factores difieren más (contraste fuerte). *Blocks* además introduce una reentrancia geométrica en los vóxeles — una traslación constante de cada cara hacia el interior.

16. Audio: frecuencias y la escala musical

Los sonidos de la interfaz se sintetizan en tiempo real vía WebAudio. Cada sonido es una **onda senoidal pura**:

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

Los valores de f (en Hz) elegidos por el proyecto no son arbitrarios — aproximan notas musicales del *temperamento igual de 12 tonos*, en el que cada semitono multiplica la frecuencia por $2^{1/12} \approx 1{,}0595$:

Sonido	Frecuencia (Hz)	Nota más cercana	Uso
click	620	≈ D#5 / E♭5	botones
hover	1100	≈ C#6	hover en escritorio
tick	1300	≈ E6	pasos del deslizador
ok	520 → 720 → 920	arpegio ascendente	reset / descarga
pop	680 + 980	intervalo de quinta justa	alternancias
error	200	≈ G#3	acción inválida

Quinta justa (razón 3 : 2): el sonido *pop* mezcla 680 Hz y 980 Hz; $980/680 \approx 1{,}44$ (una quinta justa pura sería 1,5), suficiente para sonar consonante. El **arpegio del sonido *ok*** presenta razones sucesivas cercanas a terceras: 520 → 720 (razón 1,38, cercana a la tercera menor temperada de 1,2 ligeramente ensanchada) y 720 → 920, configuración que establece una cadencia ascendente percibida como conclusiva. El envolvente tiene duración entre 12 y 160 ms y ganancia máxima de 0,035 (aproximadamente −29 dBFS), nivel deliberadamente bajo para evitar fatiga auditiva en uso prolongado.

17. Conclusión

El presente documento describió, de forma sistemática, los fundamentos matemáticos empleados en Pixel Round. En aproximadamente mil líneas de código JavaScript, el proyecto integra contribuciones de diversas áreas:

- **Geometría analítica**: ecuaciones implícitas de la elipse y el elipsoide como criterio primario de pertenencia.
- **Algoritmos clásicos de rasterización**: Bresenham en la formulación de punto medio, con extensión de Pitteway/Kappel para elipses.
- **Topología digital**: operadores de borde discreto, conceptos de 4-, 6- y 26-conectividad y composición por rebanadas.
- **Cálculo integral**: volumen del elipsoide como integral triple y su contraparte discreta (medida de conteo sobre vóxeles).
- **Álgebra lineal**: planos de corte definidos por desigualdades lineales y proyección perspectiva tridimensional.
- **Trigonometría**: parametrización de la cámara en coordenadas esféricas y ajuste automático de distancia en función del FOV.
- **Síntesis de señales**: ondas senoidales puras y aproximación al temperamento igual de doce tonos.

La observación central que el estudio permite formular es la siguiente: en una rejilla discreta, **no existe una única representación correcta** de una curva continua. Cada algoritmo presentado en este documento corresponde a una definición matemática distinta del criterio de inclusión de celdas, y cada definición produce una silueta con propiedades formales propias — suavidad en el algoritmo Euclídeo, patrón escalonado de Bresenham, mayor cobertura en el criterio de cobertura por esquina. La elección del algoritmo es, por tanto, una decisión técnica que debe considerar el objetivo de representación visual pretendido y las características métricas deseadas para la forma resultante.