Pixel Round

A Matemática do Projeto

Documentação técnica e didática dos fundamentos matemáticos empregados no gerador de círculos, elipses, esferas e elipsóides em pixel art e voxel art. Para cada conceito, apresentam-se a definição formal, a justificação geométrica e a correspondência direta com o código-fonte do projeto.

Áreas abordadas: geometria analítica · rasterização discreta · topologia digital · cálculo integral · álgebra linear · trigonometria · síntese de áudio.

Sumário

1. Visão geral: natureza matemática do problema

O Pixel Round é um gerador de formas arredondadas para **pixel art** e **voxel art**: círculos e elipses em 2D, esferas e elipsóides em 3D. O problema central pertence à classe da *rasterização discreta*: dada uma curva ou superfície definida analiticamente sobre os números reais $\mathbb{R}$, determinar quais células de uma grade regular (pixels em 2D, voxels em 3D) devem ser marcadas para representá-la.

Esse problema não admite solução única. Diferentes critérios de inclusão produzem diferentes silhuetas discretas, e cada critério possui justificativa matemática própria. Por essa razão o projeto implementa três algoritmos 2D distintos, três modos de renderização e três estilos de sombreamento 3D, descritos nas seções subsequentes.

A execução ocorre integralmente no navegador, sem componente servidor, o que impõe à formulação matemática um requisito adicional de *eficiência computacional*. As áreas teóricas mobilizadas incluem:

- geometria analítica das cônicas e quádricas;
- algoritmos incrementais com aritmética inteira (Bresenham);
- topologia digital (vizinhanças 4-, 6-e 26-conectadas);
- cálculo integral e medida de contagem;
- álgebra linear (planos de corte, projeção perspectiva);
- trigonometria e coordenadas esféricas;
- síntese de sinais e acústica fundamental.

2. Sistema de coordenadas e a malha discreta de pixels

O canvas é uma grade de $G_x \times G_y$ células. Cada célula (i, j) ocupa o quadrado $[i, i+1] \times [j, j+1]$. Em código contínuo, o **centro** da célula fica em $(i+\frac{1}{2},\ j+\frac{1}{2})$. Essa convenção é crítica: comparar a circunferência à *quina* (i, j) ou ao *centro* $(i+\frac{1}{2}, j+\frac{1}{2})$ muda quais células são consideradas pertencentes à forma.

$$\text{centro da célula } (i,j) \;=\; (i+\tfrac{1}{2},\ \ j+\tfrac{1}{2})$$

Para uma forma de diâmetro D, o código expande a malha para $D+2$ células em cada eixo (margem de uma célula em cada lado). Isso garante que pixels extremos (norte, sul, leste, oeste) nunca caiam no limite da matriz. O **centro** da forma fica em $(G_x/2,\ G_y/2)$, e os *semi-eixos* são $r_x = D/2$ e $r_y = D/2$ (ou $W/2$ e $H/2$ para uma elipse).

3. A equação fundamental: círculo e elipse

Toda a teoria do projeto começa com a equação implícita da elipse centrada na origem com semi-eixos r_x e r_y:

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

Quando $r_x = r_y = r$, a elipse degenera em uma **circunferência** de raio r. Pontos no *interior* satisfazem ≤ 1; pontos no *exterior*, > 1. Essa é a desigualdade que define se um pixel pertence ou não à forma. Os três algoritmos são, no fundo, três maneiras diferentes de aplicar (ou aproximar) esse teste em uma grade.

Deslocando o centro para (c_x, c_y) e avaliando no centro de cada célula:

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

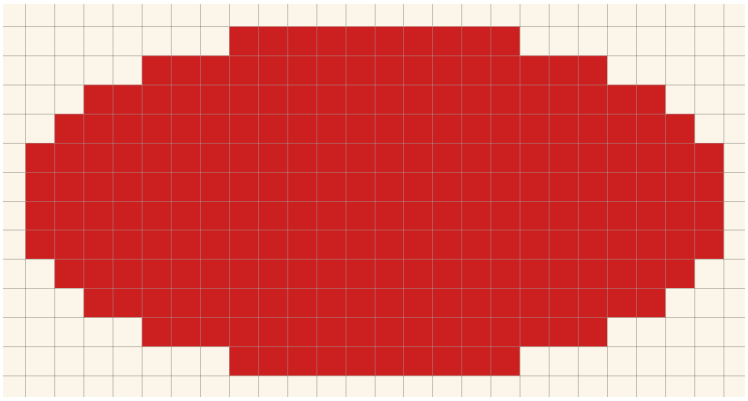

Figura 1: Elipse com $W = 24$ e $H = 12$ sobre a malha discreta.

4. Algoritmo Euclidiano (teste de distância)

Ideia. Para cada linha j, encontre analiticamente as colunas i que estão dentro da elipse. Resolvendo a equação da elipse para x, dado y:

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

Para uma linha j cujo deslocamento vertical é $d_y = j + \frac{1}{2} - c_y$, a *meia-largura horizontal* da elipse naquela altura é:

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

Se $d_y^2 / r_y^2 \geq 1$, a linha não intercepta a elipse e é pulada. Caso contrário, todas as colunas i tais que $c_x - dxM \leq i + \frac{1}{2} \leq c_x + dxM$ são preenchidas. Os limites inteiros saem por:

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

Justificação. O termo $+\frac{1}{2}$ resulta da convenção de que a célula de índice i tem centro em $i + \frac{1}{2}$. As funções $\lceil \cdot \rceil$ e $\lfloor \cdot \rfloor$ realizam a conversão do intervalo contínuo para índices inteiros, garantindo que apenas células cujo **centro** pertence ao interior da elipse sejam incluídas. Esta formulação produz a aproximação discreta mais próxima da curva contínua e, consequentemente, a silhueta de maior suavidade visual. Para diâmetros pequenos, contudo, o resultado pode apresentar menor caráter de pixel art em comparação aos demais algoritmos.

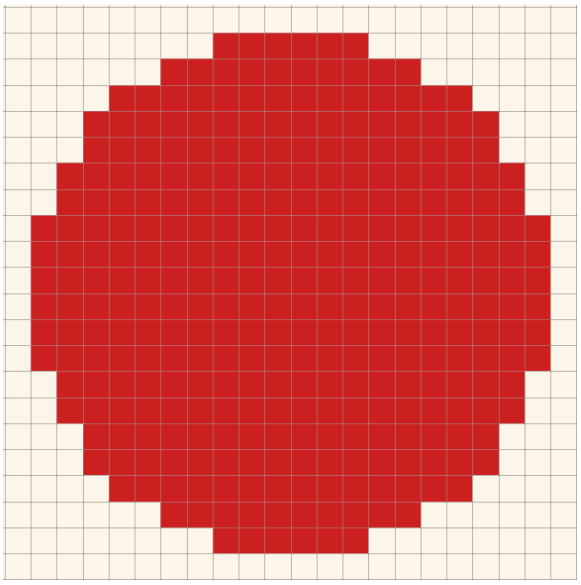

Figura 2: Algoritmo Euclidiano para $D = 20$.

5. Algoritmo de Bresenham (ponto médio inteiro)

O algoritmo de Bresenham, publicado em 1965 para o traçado de segmentos de reta, foi posteriormente estendido para circunferências e generalizado por Pitteway e Kappel para elipses arbitrárias. Sua propriedade distintiva é dispensar operações em ponto flutuante e extração de raiz quadrada: a determinação do próximo pixel utiliza exclusivamente **somas e comparações sobre inteiros**, por meio de uma *função de decisão* que avalia em qual lado da curva analítica se encontra o ponto médio entre os dois pixels candidatos.

5.1 Caso circular

Para uma circunferência de raio inteiro R, parte-se de $(x{=}0,\ y{=}R)$ com parâmetro de decisão inicial:

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

A cada passo (no octante de 0 a 45°):

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

Justificativa. A função d aproxima, a cada iteração, o valor da equação implícita da circunferência avaliado no **ponto médio** entre as duas opções de pixel candidato:

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

O sinal de d indica se o ponto médio se encontra no interior da circunferência (caso em que se avança apenas na horizontal) ou no exterior (caso em que se decrementa também a coordenada vertical). Os oito octantes da circunferência são obtidos por aplicação das simetrias do grupo diedral D_4 ao octante computado, o que corresponde, no código, às oito chamadas a `span(...)`. O traçado resultante exibe o padrão escalonado característico das aproximações discretas de curvas suaves.

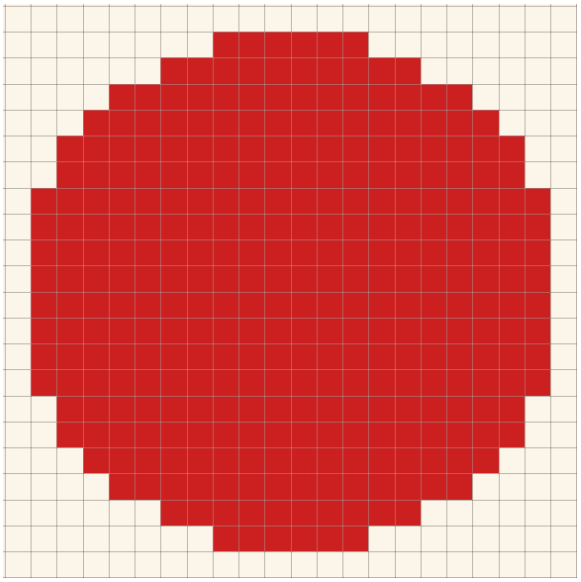

Figura 3: Algoritmo de Bresenham para $D = 20$.

5.2 Caso elíptico (duas regiões)

Para uma elipse com semi-eixos a, b, o algoritmo divide o primeiro quadrante em **duas regiões**, delimitadas pelo ponto onde a tangente faz 45°:

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

Os parâmetros de decisão iniciais são:

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

Em cada iteração, p é atualizado por incrementos pré-calculados (todos inteiros, após multiplicar por 4). O resultado é o mínimo de aritmética possível por pixel: *um teste de sinal e duas somas*. O código usa `Math.floor` ao calcular a e b a partir de r_x e r_y para evitar adicionar uma linha extra fora do *bounding-box* quando r_y é não-inteiro.

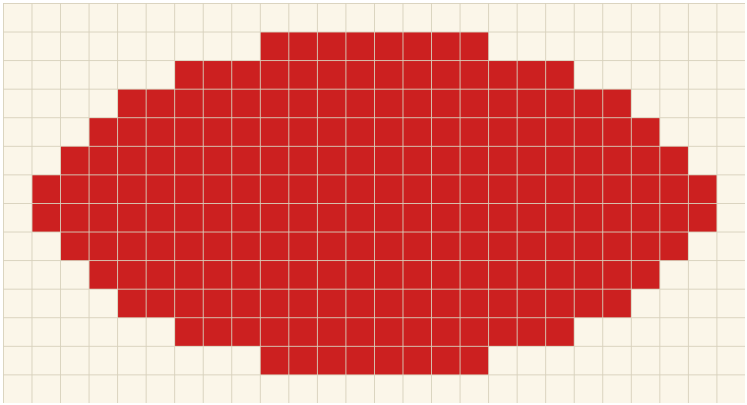

Figura 4: Algoritmo de Bresenham aplicado à elipse $W = 24$, $H = 12$.

6. Algoritmo de Limiar (Threshold / cobertura de quina)

Este algoritmo pergunta: *existe alguma quina da célula que esteja dentro da elipse?* Para cada célula (i, j), o código procura a quina mais próxima do centro da forma (projeção do centro nas arestas da célula):

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

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

O valor 0,94 é um *limiar empírico* ligeiramente inferior a 1,0. Sua função é eliminar o artefato conhecido como *spike tangencial* de 1 pixel, que ocorre nos quatro pontos cardeais (N, S, L, O), onde a curva é tangente a uma aresta inteira da célula. Sem essa redução, o algoritmo incluiria uma célula adicional cuja contribuição não melhora a representação visual da forma.

Entre os três algoritmos, este é o que produz a silhueta de maior área para um mesmo diâmetro D, uma vez que a condição de inclusão é a menos restritiva: basta que uma única quina da célula satisfaça a desigualdade da elipse para que a célula inteira seja preenchida.

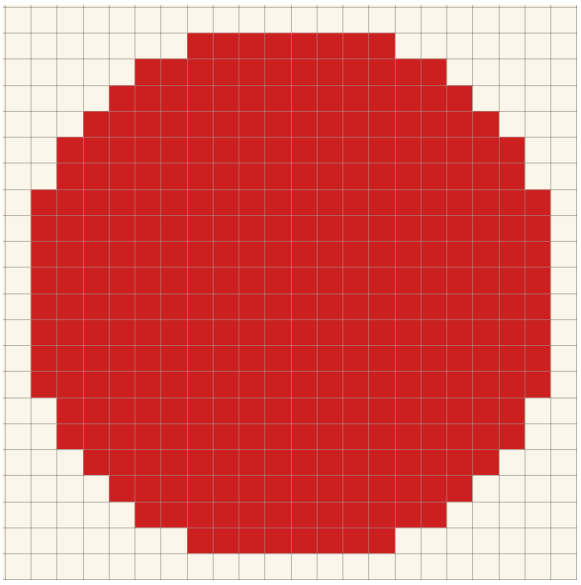

Figura 5: Algoritmo de Limiar para $D = 20$.

7. Modos de renderização: Preenchido, Fino, Grosso

Tendo uma matriz $f[j][i] = 1$ para células internas, o projeto deriva três estilos visuais por **topologia discreta**:

7.1 Preenchido

Retorna f sem modificação. Todos os pixels internos.

7.2 Fino (outline)

Um pixel pertence ao contorno fino se está preenchido *e* tem pelo menos um vizinho ortogonal (N, S, L, O) *fora* da forma. Em notação lógica:

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

Geometricamente: é o **bordo discreto** da forma, análogo discreto da fronteira topológica $\partial F = F \cap \overline{F^{\mathrm{c}}}$.

7.3 Grosso (thick)

O modo grosso adiciona aos pixels de borda os **pontes diagonais**: células internas que têm simultaneamente um vizinho-borda horizontal *e* um vizinho-borda vertical. Isso fecha as diagonais de 1 pixel que o modo fino deixa abertas, útil

quando o contorno precisa parecer estável num cenário (sem buracos 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

A função `area2D` simplesmente **conta** as células marcadas em f. Isso é a *medida de contagem*, que aproxima a integral de Riemann da função indicadora da elipse:

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

Para D grande, área discreta/área contínua $\to 1$. Para D pequeno, a diferença entre algoritmos é visível: Threshold tende a superestimar; Euclidiano fica próximo da área ideal; Bresenham fica em algum lugar entre os dois.

9. Da 2D para 3D: a equação do elipsóide

Em 3D, a forma fundamental é o **elipsóide** centrado em (c_x, c_y, c_z) com semi-eixos r_x, r_y, r_z. Sua equação é a generalização natural da elipse:

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

Quando $r_x = r_y = r_z$, voltamos à **esfera**. Avaliando no centro de cada voxel:

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

10. Voxelização e cálculo de volume

O volume contínuo do elipsóide é um resultado clássico do cálculo integral, obtido por integração em discos:

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

A versão discreta percorre cada voxel da caixa $D_x \times D_y \times D_z$ e incrementa um contador quando $a_x^2 + a_y^2 + a_z^2 \leq 1$. Esse é o algoritmo de `voxelVolume`.

Casca (shell) vs. interior. Para o modo *filled* o projeto mostra apenas os voxels com pelo menos um vizinho-6 fora do conjunto preservado (ou seja, voxels visíveis da superfície externa ou da face de corte). Para *thin*, mostra apenas a **superfície** do elipsóide completo (1 voxel de espessura).

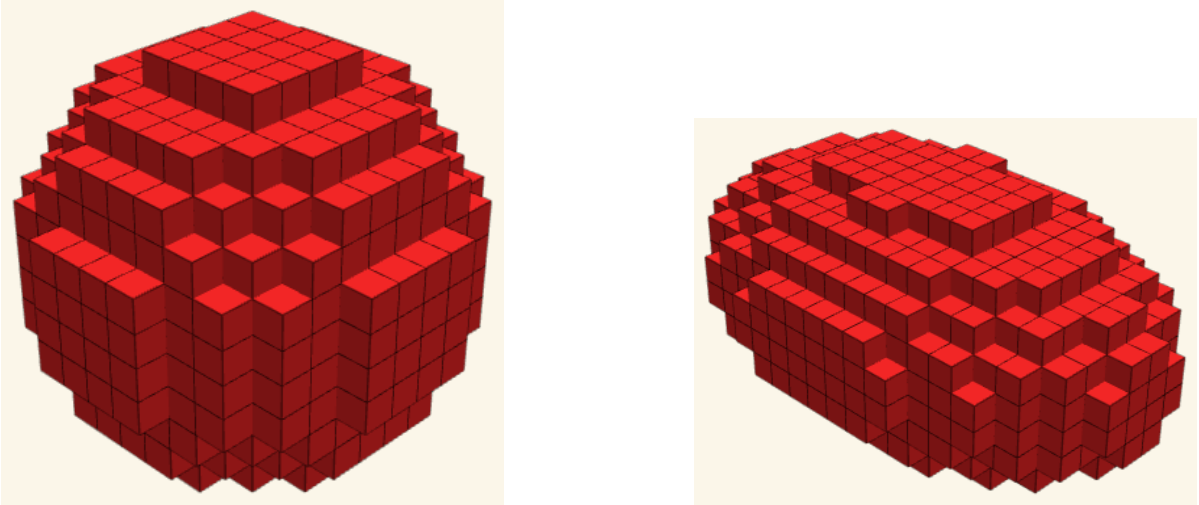

Figura 6: Esfera ($D = 12$) e elipsoide ($W = 20$, $H = 10$, $D = 12$) voxelizados.

11. Cortes geométricos: planos e o corte diagonal a 45°

O Pixel Round permite cortar o sólido por **três planos**. Cada corte é uma *desigualdade linear* que descarta voxels:

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

Os planos X e Y são perpendiculares aos eixos coordenados, com vetores normais $(1, 0, 0)$ e $(0, 1, 0)$. O *plano diagonal* tem vetor normal $(1, 1, 0)/\sqrt{2}$ e corta a 45° no plano XY: matematicamente, é a família de planos $x+y = k$. Como o máximo de $x + y$ em uma caixa $D_x \times D_y$ vale $D_x + D_y$, o slider do corte diagonal tem amplitude $D_x + D_y$, enquanto X e Y têm amplitudes D_x e D_y, respectivamente. O corte é **proporcional**: o código guarda uma porcentagem $cutPct$ e recalcula o limite como

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

de modo que 50% num eixo continua sendo 50% quando o usuário troca de eixo ou redimensiona.

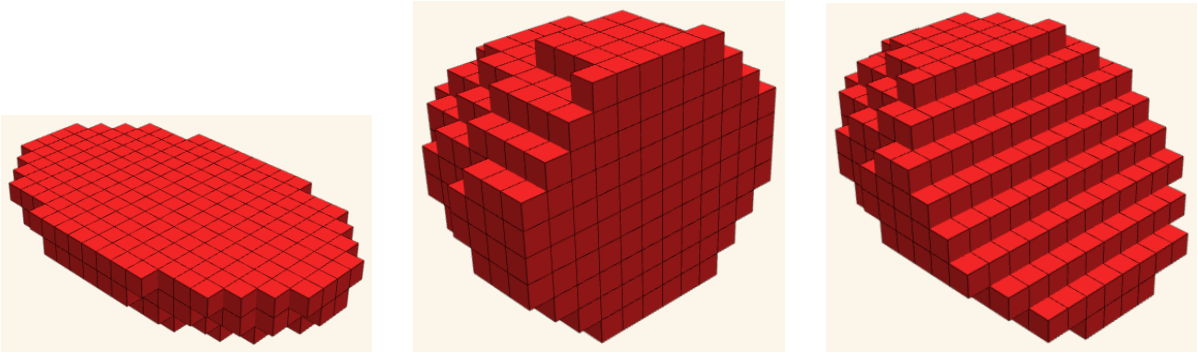

Figura 7: Cortes planos a 50% nos eixos Y, X e diagonal (da esquerda para a direita).

12. Modo Grosso 3D: tridimensionalização por fatias

Para o modo *thick* em 3D, o projeto aplica o algoritmo grosso 2D em **todas as fatias** nos três eixos: XY para cada z, XZ para cada y, YZ para cada x. Em seguida faz a *união* dos resultados. A motivação geométrica é a estanqueidade: o conjunto mínimo de voxels que tampam todas as diagonais sem duplicar a espessura da casca. Formalmente, sendo $S_z(z)$ a fatia XY na altura z e T o operador grosso 2D:

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

O resultado é depois interseccionado com o conjunto preservado pelo corte. A teoria por trás é **topologia digital**: o operador grosso garante *26-conectividade* da casca (vizinhos incluindo diagonais), útil para construção em Minecraft onde voxels diagonais não se tocam por face.

13. Câmera 3D: coordenadas esféricas

A câmera 3D orbita o objeto a uma distância r com dois ângulos — θ (azimute, no plano horizontal) e φ (polar, do eixo vertical). Essa é a **convenção física** de coordenadas esféricas, e a conversão para cartesianas (a forma que *three.js* entende) é:

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

Posição inicial: $\theta = \pi/4$ (45°, perspectiva isométrica) e $\varphi = \pi/3$ (60° a partir do polo Norte, levemente acima do equador). Arrasto do mouse incrementa θ e φ diretamente; roda/pinça incrementa r. A simplicidade dessa parametrização é o que permite a rotação livre sem nenhum quatérnio ou matriz explícita.

14. Auto-zoom: esfera envolvente e FOV

Quando o usuário muda o tamanho da figura ou reseta a câmera, o projeto recalcula a **distância ideal** da câmera para que o objeto caiba na tela em qualquer ângulo. A ideia é encerrar o objeto numa *esfera de raio* R (a metade da diagonal da AABB, axis-aligned bounding box):

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

Uma esfera tem **projeção invariante por rotação**, então se ela couber na tela em uma orientação, cabe em todas. Dado o FOV vertical da câmera ($fov_V = 35°$ em radianos), a distância necessária para enquadrar uma esfera de raio R é:

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

O FOV horizontal sai da razão de aspecto do canvas ($aspect$ = largura/altura) por:

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

A distância final é o máximo entre $dist_V$ e $dist_H$, multiplicado por 1,20 (20% de margem visual). Quando a figura está cortada, o código encolhe D_x ou D_y para o tamanho restante, de modo que uma esfera cortada a 75% não apareça minúscula no canto.

15. Tons (shading) e aritmética de cor RGB

Os três estilos 3D (*Classic*, *Smooth*, *Blocks*) diferenciam-se por **fatores multiplicativos** aplicados às três faces visíveis de cada voxel (topo, lado iluminado, lado escuro). A função `shadeHex(hex, factor)` parseia o hex em (R, G, B) e multiplica cada canal, com *clamp* em $[0, 255]$:

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

Isso é uma *aproximação linear* da função de iluminação Lambertiana ideal,

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

quando os três fatores correspondem ao cosseno do ângulo entre a normal de cada face e a direção de luz fixa. Em *Smooth* os fatores são próximos (faces parecidas); em *Classic* os fatores diferem mais (contraste forte). *Blocks* ainda introduz uma reentrância geométrica nos voxels — uma translação constante de cada face para dentro.

16. Áudio: frequências e a escala musical

Os sons da interface são sintetizados em tempo real via WebAudio. Cada som é uma **onda senoidal pura**:

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

Os valores de f (em Hz) escolhidos pelo projeto não são aleatórios — aproximam notas musicais do *temperamento igual de 12 tons*, em que cada semitom multiplica a frequência por $2^{1/12} \approx 1{,}0595$:

Som	Frequência (Hz)	Nota mais próxima	Uso
click	620	≈ D#5 / E♭5	botões
hover	1100	≈ C#6	hover desktop
tick	1300	≈ E6	passos do slider
ok	520 → 720 → 920	arpejo ascendente	reset / download
pop	680 + 980	intervalo de quinta justa	toggles
error	200	≈ G#3	ação inválida

Quinta justa (razão 3 : 2): o som *pop* mistura 680 Hz e 980 Hz; 980/680 ≈ 1,44 (a quinta justa pura seria 1,5), suficiente para soar consonante. O **arpejo do som *ok*** apresenta razões sucessivas próximas a terças: 520 → 720 (razão 1,38, próxima à terça menor temperada de 1,2 ligeiramente alargada) e 720 → 920, configuração que estabelece uma cadência ascendente percebida como conclusiva. O envelope tem duração entre 12 e 160 ms e ganho máximo de 0,035 (aproximadamente −29 dBFS), nível deliberadamente baixo para evitar fadiga auditiva em uso prolongado.

17. Conclusão

O presente documento descreveu, de forma sistemática, os fundamentos matemáticos empregados no Pixel Round. Em aproximadamente mil linhas de código JavaScript, o projeto integra contribuições de diversas áreas:

- **Geometria analítica**: equações implícitas da elipse e do elipsóide como critério primário de pertinência.
- **Algoritmos clássicos de rasterização**: Bresenham na formulação de ponto médio, com extensão de Pitteway/Kappel para elipses.
- **Topologia digital**: operadores de bordo discreto, conceitos de 4-, 6-e 26--conectividade e composição por fatias.
- **Cálculo integral**: volume do elipsóide como integral tripla e sua contraparte discreta (medida de contagem sobre voxels).
- **Álgebra linear**: planos de corte definidos por desigualdades lineares e projeção perspectiva tridimensional.
- **Trigonometria**: parametrização da câmera em coordenadas esféricas e ajuste automático de distância em função do FOV.
- **Síntese de sinais**: ondas senoidais puras e aproximação ao temperamento igual de doze tons.

A observação central que o estudo permite formular é a seguinte: em uma malha discreta, **não existe uma única representação correta** de uma curva contínua. Cada algoritmo apresentado neste documento corresponde a uma definição matemática distinta do critério de inclusão de células, e cada definição produz uma silhueta com propriedades formais próprias — suavidade no algoritmo Euclidiano, padrão escalonado de Bresenham, maior abrangência no critério de cobertura por quina. A escolha do algoritmo, portanto, é uma decisão técnica que deve considerar o objetivo de representação visual pretendido e as características métricas desejadas para a forma resultante.