Pixel Round

The Mathematics of the Project

Technical and didactic documentation of the mathematical foundations employed in the generator of circles, ellipses, spheres and ellipsoids for pixel art and voxel art. For each concept, the formal definition, the geometric justification and the direct correspondence with the project's source code are presented.

Topics covered: analytic geometry · discrete rasterization · digital topology · integral calculus · linear algebra · trigonometry · audio synthesis.

Table of Contents

1 Overview: mathematical nature of the problem 3

2 Coordinate system and the discrete pixel grid 3

3 The fundamental equation: circle and ellipse 4

4 Euclidean algorithm (distance test) 4

5 Bresenham algorithm (integer midpoint) 6
- 5.1 Circular case . . . 6
- 5.2 Elliptic case (two regions) . . . 7

6 Threshold algorithm (corner coverage) 8

7 Rendering modes: Filled, Thin, Thick 9
- 7.1 Filled . . . 9
- 7.2 Thin (outline) . . . 9
- 7.3 Thick . . . 9

8 Discrete area computation 10

9 From 2D to 3D: the ellipsoid equation 10

10 Voxelization and volume computation 10

11 Geometric cuts: planes and the $45°$ diagonal cut 11

12 Thick 3D mode: tridimensionalization by slices 12

13 3D camera: spherical coordinates 12

14 Auto-zoom: bounding sphere and FOV 13

15 Shading and RGB color arithmetic 13

16 Audio: frequencies and the musical scale 14

17 Conclusion 14

1. Overview: mathematical nature of the problem

Pixel Round is a generator of rounded shapes for **pixel art** and **voxel art**: circles and ellipses in 2D, spheres and ellipsoids in 3D. The central problem belongs to the class of *discrete rasterization*: given a curve or surface defined analytically over the real numbers $\mathbb{R}$, determine which cells of a regular grid (pixels in 2D, voxels in 3D) must be marked to represent it.

This problem admits no unique solution. Different inclusion criteria produce different discrete silhouettes, and each criterion has its own mathematical justification. For this reason the project implements three distinct 2D algorithms, three rendering modes and three 3D shading styles, all described in the subsequent sections.

Execution occurs entirely in the browser, with no server component, which imposes on the mathematical formulation an additional requirement of *computational efficiency*. The theoretical areas involved include:

- analytic geometry of conics and quadrics;
- incremental algorithms with integer arithmetic (Bresenham);
- digital topology (4-, 6- and 26-connected neighborhoods);
- integral calculus and counting measure;
- linear algebra (cutting planes, perspective projection);
- trigonometry and spherical coordinates;
- signal synthesis and fundamental acoustics.

2. Coordinate system and the discrete pixel grid

The canvas is a grid of $G_x \times G_y$ cells. Each cell (i, j) occupies the square $[i, i+1] \times [j, j+1]$. In continuous code, the **center** of the cell is at $(i + \frac{1}{2},\ j + \frac{1}{2})$. This convention is critical: comparing the circumference to the *corner* (i, j) or to the *center* $(i + \frac{1}{2}, j + \frac{1}{2})$ changes which cells are considered as belonging to the shape.

$$\text{center of cell } (i, j) \ = \ (i + \tfrac{1}{2},\ \ j + \tfrac{1}{2})$$

For a shape of diameter D, the code expands the grid to $D+2$ cells along each axis (a margin of one cell on each side). This ensures that extreme pixels (N, S, E, W) never fall on the matrix boundary. The **center** of the shape is at $(G_x/2,\ G_y/2)$,

and the *semi-axes* are $r_x = D/2$ and $r_y = D/2$ (or $W/2$ and $H/2$ for an ellipse).

3. The fundamental equation: circle and ellipse

All theory in the project begins with the implicit equation of the ellipse centered at the origin with semi-axes r_x and r_y:

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

When $r_x = r_y = r$, the ellipse degenerates into a **circumference** of radius r. Points in the *interior* satisfy ≤ 1; points in the *exterior*, > 1. This is the inequality that determines whether a pixel belongs to the shape. The three algorithms are, fundamentally, three different ways of applying (or approximating) this test on a grid.

Shifting the center to (c_x, c_y) and evaluating at the center of each cell:

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

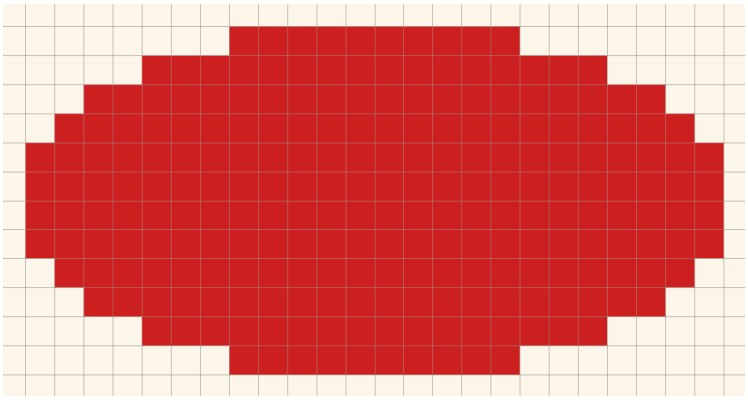

Figure 1: Ellipse with $W = 24$ and $H = 12$ on the discrete grid.

4. Euclidean algorithm (distance test)

Idea. For each row j, analytically find the columns i that lie inside the ellipse. Solving the ellipse equation for x, given y:

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

For a row j whose vertical displacement is $d_y = j + \frac{1}{2} - c_y$, the *horizontal half-width* of the ellipse at that height is:

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

If $d_y^2 / r_y^2 \geq 1$, the row does not intersect the ellipse and is skipped. Otherwise, every column i such that $c_x - dxM \leq i + \frac{1}{2} \leq c_x + dxM$ is filled. The integer bounds come from:

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

Justification. The $+\frac{1}{2}$ term arises from the convention that cell i has center at $i + \frac{1}{2}$. The functions $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$ perform the conversion from a continuous interval to integer indices, ensuring that only cells whose **center** lies in the interior of the ellipse are included. This formulation yields the discrete approximation closest to the continuous curve and, consequently, the silhouette of greatest visual smoothness. For small diameters, however, the result may exhibit a less pronounced pixel-art character compared to the other algorithms.

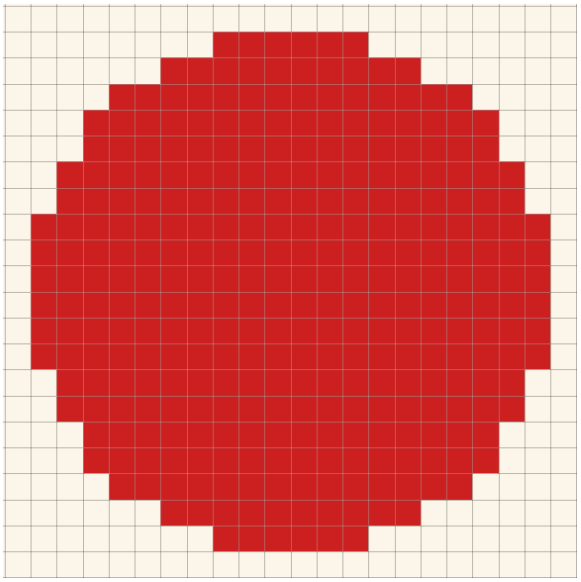

Figure 2: Euclidean algorithm at $D = 20$.

5. Bresenham algorithm (integer midpoint)

The Bresenham algorithm, published in 1965 for drawing line segments, was later extended to circles and generalized by Pitteway and Kappel to arbitrary ellipses. Its distinguishing property is to dispense with floating-point operations and square root extraction: determining the next pixel uses exclusively **integer additions and comparisons**, by means of a *decision function* that evaluates on which side of the analytic curve the midpoint between the two candidate pixels lies.

5.1 Circular case

For a circle of integer radius R, the algorithm starts at (x=0, y=R) with the initial decision parameter:

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

At each step (in the octant from 0 to 45°):

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

Justification. The function d approximates, at each iteration, the value of the implicit circle equation evaluated at the **midpoint** between the two candidate pixel options:

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

The sign of d indicates whether the midpoint lies inside the circle (in which case one advances only horizontally) or outside (in which case the vertical coordinate is also decremented). The eight octants of the circle are obtained by applying the symmetries of the dihedral group D_4 to the computed octant, which corresponds, in code, to the eight `span(...)` calls. The resulting trace exhibits the stepped pattern characteristic of discrete approximations of smooth curves.

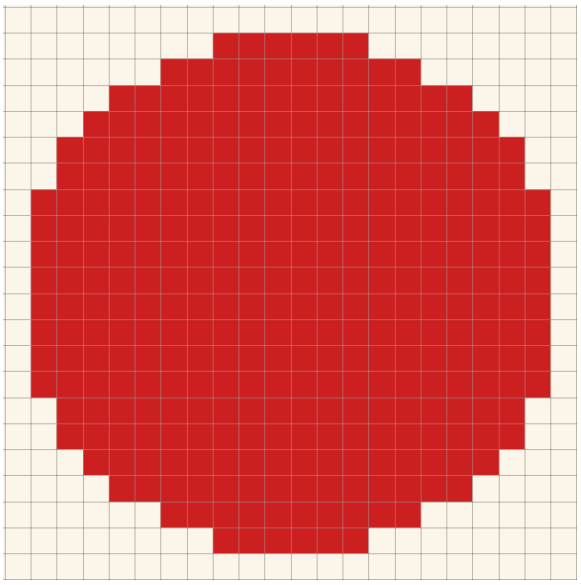

Figure 3: Bresenham algorithm at $D = 20$.

5.2 Elliptic case (two regions)

For an ellipse with semi-axes a, b, the algorithm divides the first quadrant into **two regions**, delimited by the point where the tangent makes 45°:

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

The initial decision parameters are:

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

At each iteration, p is updated by precomputed increments (all integers, after multiplying by 4). The result is the minimum possible arithmetic per pixel: *one sign test and two additions*. The code uses `Math.floor` when computing a and b from r_x and r_y to avoid adding an extra row outside the bounding box when r_y is non-integer.

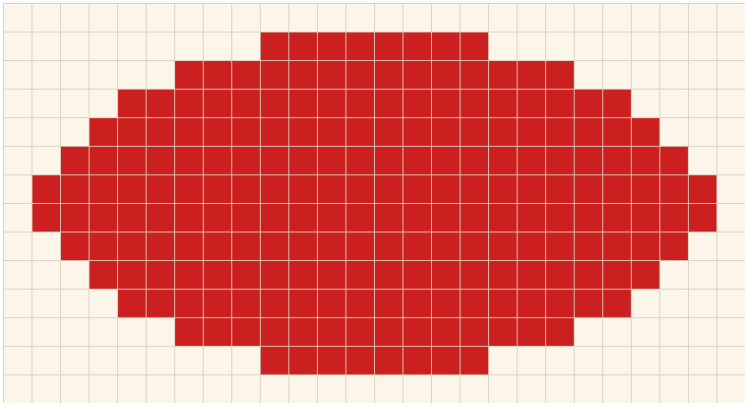

Figure 4: Bresenham algorithm applied to the ellipse $W = 24$, $H = 12$.

6. Threshold algorithm (corner coverage)

This algorithm asks: *is there any corner of the cell inside the ellipse?* For each cell (i, j), the code finds the corner closest to the center of the shape (projection of the center onto the cell edges):

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

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

The value 0,94 is an *empirical threshold* slightly below 1,0. Its purpose is to eliminate the artifact known as the *tangential spike* of 1 pixel that occurs at the four cardinal points (N, S, E, W), where the curve is tangent to a full edge of the cell. Without this reduction, the algorithm would include an additional cell whose contribution does not improve the visual representation of the shape.

Among the three algorithms, this is the one that produces the silhouette of largest area for the same diameter D, since the inclusion condition is the least restrictive: a single corner of the cell satisfying the ellipse inequality suffices to fill the entire cell.

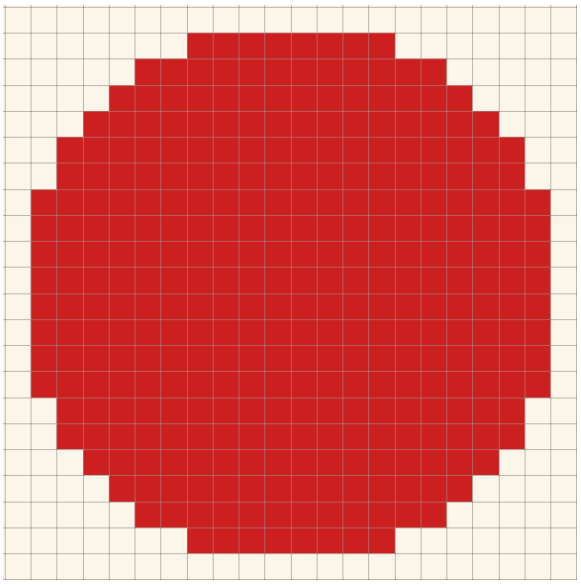

Figure 5: Threshold algorithm at $D = 20$.

7. Rendering modes: Filled, Thin, Thick

Having a matrix $f[j][i] = 1$ for interior cells, the project derives three visual styles by **discrete topology**:

7.1 Filled

Returns f unchanged. All interior pixels.

7.2 Thin (outline)

A pixel belongs to the thin contour if it is filled *and* has at least one orthogonal neighbor (N, S, E, W) *outside* the shape. In logical notation:

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

Geometrically: this is the **discrete boundary** of the shape, the discrete analog of the topological boundary $\partial F = F \cap \overline{F^{c}}$.

7.3 Thick

The thick mode adds to the boundary pixels the **diagonal bridges**: interior cells that have simultaneously a horizontal and a vertical boundary neighbor. This closes the 1-pixel diagonals that thin mode leaves open, useful when the contour

must look stable in a scene (no 45° holes).

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

8. Discrete area computation

The `area2D` function simply **counts** the cells marked in f. This is the *counting measure*, which approximates the Riemann integral of the indicator function of the ellipse:

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

For large D, discrete area/continuous area $\to 1$. For small D, the difference between algorithms is visible: Threshold tends to overestimate; Euclidean stays close to the ideal area; Bresenham falls somewhere between the two.

9. From 2D to 3D: the ellipsoid equation

In 3D, the fundamental shape is the **ellipsoid** centered at (c_x, c_y, c_z) with semi-axes r_x, r_y, r_z. Its equation is the natural generalization of the ellipse:

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

When $r_x = r_y = r_z$, we recover the **sphere**. Evaluating at the center of each voxel:

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

10. Voxelization and volume computation

The continuous volume of the ellipsoid is a classical result of integral calculus, obtained by integration over disks:

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

The discrete version iterates over every voxel of the box $D_x \times D_y \times D_z$ and increments a counter when $a_x^2 + a_y^2 + a_z^2 \leq 1$. This is the `voxelVolume` algorithm.

Shell vs. interior. For the *filled* mode, the project displays only voxels with at least one 6-neighbor outside the preserved set (that is, voxels visible from the external surface or from the cut face). For *thin*, only the **surface** of the complete ellipsoid is shown (1-voxel thickness).

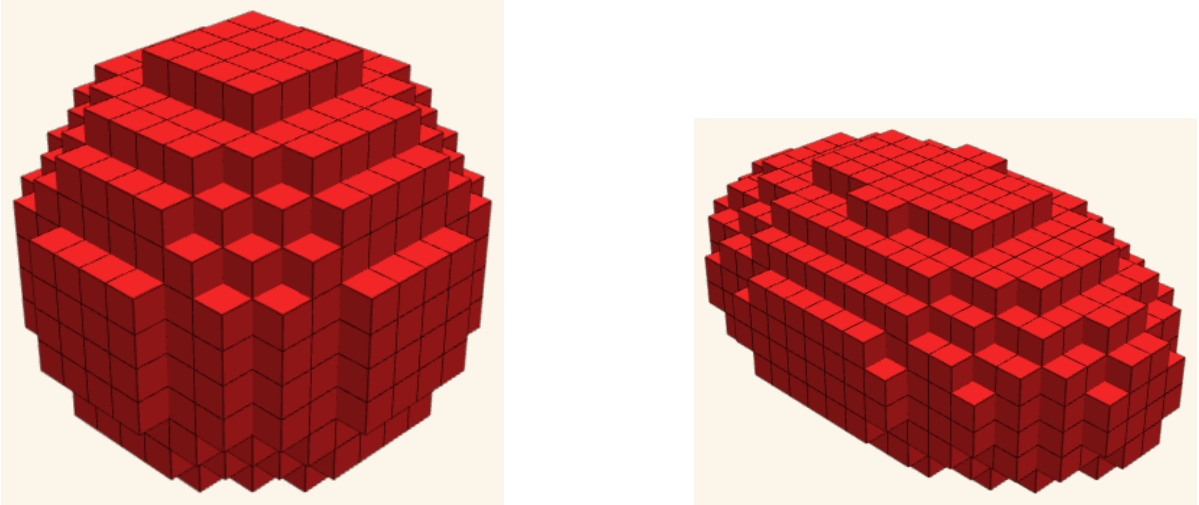

Figure 6: Voxelised sphere ($D = 12$) and ellipsoid ($W = 20$, $H = 10$, $D = 12$).

11. Geometric cuts: planes and the 45° diagonal cut

Pixel Round allows cutting the solid by **three planes**. Each cut is a *linear inequality* that discards voxels:

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

The X and Y planes are perpendicular to the coordinate axes, with normal vectors $(1, 0, 0)$ and $(0, 1, 0)$. The *diagonal plane* has normal vector $(1, 1, 0)/\sqrt{2}$ and cuts at 45° in the XY plane: mathematically, this is the family of planes $x + y = k$. Since the maximum of $x + y$ in a box $D_x \times D_y$ is $D_x + D_y$, the diagonal cut slider has range $D_x + D_y$, while X and Y have ranges D_x and D_y, respectively. The cut is **proportional**: the code stores a percentage *cutPct* and recomputes the limit as

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

so 50% on one axis remains 50% when the user switches axis or resizes.

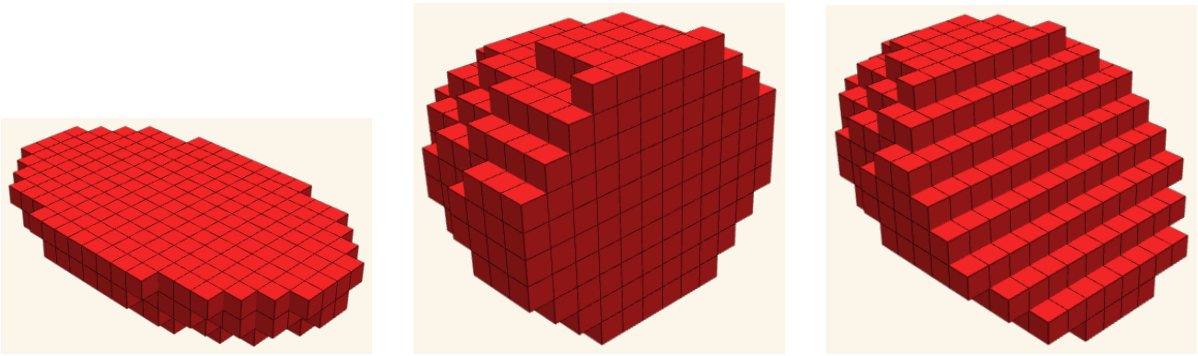

Figure 7: Planar cuts at 50% along Y, X and the diagonal axis (left to right).

12. Thick 3D mode: tridimensionalization by slices

For *thick* mode in 3D, the project applies the 2D thick algorithm to **all slices** along the three axes: XY for each z, XZ for each y, YZ for each x. It then takes the *union* of the results. The geometric motivation is watertightness: the minimum set of voxels that plug all the diagonals without doubling the shell thickness. Formally, with $S_z(z)$ the XY slice at height z and T the 2D thick operator:

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

The result is then intersected with the set preserved by the cut. The underlying theory is **digital topology**: the thick operator guarantees *26-connectivity* of the shell (neighbors including diagonals), useful for construction in Minecraft where diagonally adjacent voxels do not touch by face.

13. 3D camera: spherical coordinates

The 3D camera orbits the object at a distance r with two angles — θ (azimuth, in the horizontal plane) and φ (polar, from the vertical axis). This is the **physical convention** of spherical coordinates, and the conversion to Cartesian (the form three.js expects) is:

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

Initial position: $\theta = \pi/4$ (45°, isometric perspective) and $\varphi = \pi/3$ (60° from the north pole, slightly above the equator). Mouse drag increments θ and φ directly; wheel/pinch increments r. The simplicity of this parametrization is what allows free rotation without quaternions or explicit matrices.

14. Auto-zoom: bounding sphere and FOV

When the user changes the figure size or resets the camera, the project recomputes the **ideal distance** of the camera so that the object fits the viewport at any angle. The idea is to enclose the object in a *sphere of radius* R (half the diagonal of the AABB, axis-aligned bounding box):

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

A sphere has **projection invariant under rotation**, so if it fits in the viewport at one orientation, it fits in all. Given the vertical FOV of the camera ($fov_V = 35°$ in radians), the distance required to frame a sphere of radius R is:

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

The horizontal FOV follows from the canvas aspect ratio ($aspect = \text{width}/\text{height}$) by:

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

The final distance is the maximum of $dist_V$ and $dist_H$, multiplied by 1,20 (20% visual margin). When the figure is cut, the code shrinks D_x or D_y to the remaining size, so that a sphere cut at 75% does not appear tiny in a corner.

15. Shading and RGB color arithmetic

The three 3D styles (*Classic*, *Smooth*, *Blocks*) differ by **multiplicative factors** applied to the three visible faces of each voxel (top, lit side, dark side). The `shadeHex(hex, factor)` function parses the hex into (R, G, B) and multiplies each channel, with clamping to $[0, 255]$:

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

This is a *linear approximation* of the ideal Lambertian illumination function,

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

where the three factors correspond to the cosine of the angle between the normal of each face and a fixed light direction. In *Smooth* the factors are close (similar faces); in *Classic* the factors differ more (strong contrast). *Blocks* additionally introduces a geometric inset to the voxels — a constant translation of each face inward.

16. Audio: frequencies and the musical scale

The interface sounds are synthesized in real time via WebAudio. Each sound is a **pure sine wave**:

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

The values of f (in Hz) chosen by the project are not arbitrary — they approximate musical notes of the *12-tone equal temperament*, in which each semitone multiplies the frequency by $2^{1/12} \approx 1{,}0595$:

Sound	Frequency (Hz)	Nearest note	Use
click	620	≈ D#5 / E♭5	buttons
hover	1100	≈ C#6	desktop hover
tick	1300	≈ E6	slider steps
ok	520 → 720 → 920	ascending arpeggio	reset / download
pop	680 + 980	perfect fifth interval	toggles
error	200	≈ G#3	invalid action

Perfect fifth (ratio 3 : 2): the *pop* sound mixes 680 Hz and 980 Hz; 980/680 ≈ 1,44 (a pure perfect fifth would be 1,5), enough to sound consonant. The ***ok* arpeggio** presents successive ratios close to thirds: 520 → 720 (ratio 1,38, near the tempered minor third of 1,2 slightly widened) and 720 → 920, a configuration that establishes an ascending cadence perceived as conclusive. The envelope has duration between 12 and 160 ms and peak gain of 0,035 (approximately −29 dBFS), a level deliberately low to avoid auditory fatigue in prolonged use.

17. Conclusion

This document described, in a systematic way, the mathematical foundations employed in Pixel Round. In approximately one thousand lines of JavaScript code, the project integrates contributions from several areas:

- **Analytic geometry**: implicit equations of the ellipse and ellipsoid as the primary criterion of belonging.
- **Classical rasterization algorithms**: Bresenham in the midpoint formulation, with the Pitteway/Kappel extension for ellipses.
- **Digital topology**: discrete boundary operators, concepts of 4-, 6- and 26-connectivity, and slice composition.

- **Integral calculus**: volume of the ellipsoid as a triple integral and its discrete counterpart (counting measure over voxels).
- **Linear algebra**: cutting planes defined by linear inequalities and three-dimensional perspective projection.
- **Trigonometry**: parametrization of the camera in spherical coordinates and automatic distance adjustment as a function of FOV.
- **Signal synthesis**: pure sine waves and approximation to the 12-tone equal temperament.

The central observation that the study allows to formulate is the following: on a discrete grid, **there is no single correct representation** of a continuous curve. Each algorithm presented in this document corresponds to a distinct mathematical definition of the cell-inclusion criterion, and each definition produces a silhouette with its own formal properties — smoothness in the Euclidean algorithm, the stepped pattern of Bresenham, larger coverage in the corner-coverage criterion. The choice of algorithm is therefore a technical decision that must consider the intended visual representation and the desired metric characteristics of the resulting shape.