Pixel Round

Les Mathématiques du Projet

Documentation technique et didactique des fondements mathématiques employés dans le générateur de cercles, ellipses, sphères et ellipsoïdes pour le pixel art et le voxel art. Pour chaque concept sont présentés la définition formelle, la justification géométrique et la correspondance directe avec le code source du projet.

Domaines abordés : géométrie analytique · rastérisation discrète · topologie numérique · calcul intégral · algèbre linéaire · trigonométrie · synthèse audio.

Sommaire

1. Vue d'ensemble : nature mathématique du problème

Pixel Round est un générateur de formes arrondies pour le **pixel art** et le **voxel art** : cercles et ellipses en 2D, sphères et ellipsoïdes en 3D. Le problème central appartient à la classe de la *rastérisation discrète* : étant donné une courbe ou une surface définie analytiquement sur les nombres réels $\mathbb{R}$, déterminer quelles cellules d'une grille régulière (pixels en 2D, voxels en 3D) doivent être marquées pour la représenter.

Ce problème n'admet pas de solution unique. Différents critères d'inclusion produisent différentes silhouettes discrètes, et chaque critère possède sa propre justification mathématique. C'est pour cette raison que le projet implémente trois algorithmes 2D distincts, trois modes de rendu et trois styles d'ombrage 3D, décrits dans les sections suivantes.

L'exécution a lieu intégralement dans le navigateur, sans composant serveur, ce qui impose à la formulation mathématique une exigence supplémentaire d'*efficacité computationnelle*. Les domaines théoriques mobilisés sont :

- géométrie analytique des coniques et quadriques ;
- algorithmes incrémentaux à arithmétique entière (Bresenham) ;
- topologie numérique (voisinages 4-, 6- et 26-connexes) ;
- calcul intégral et mesure de comptage ;
- algèbre linéaire (plans de coupe, projection perspective) ;
- trigonométrie et coordonnées sphériques ;
- synthèse de signaux et acoustique fondamentale.

2. Système de coordonnées et la grille discrète de pixels

Le canevas est une grille de $G_x \times G_y$ cellules. Chaque cellule (i, j) occupe le carré $[i, i+1] \times [j, j+1]$. En code continu, le **centre** de la cellule se trouve en $(i + \frac{1}{2},\ j + \frac{1}{2})$. Cette convention est cruciale : comparer la circonférence au *coin* (i, j) ou au *centre* $(i + \frac{1}{2}, j + \frac{1}{2})$ change quelles cellules sont considérées comme appartenant à la forme.

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

Pour une forme de diamètre D, le code étend la grille à $D+2$ cellules sur chaque axe (marge d'une cellule de chaque côté). Cela garantit que les pixels extrêmes (N, S, E, O) ne tombent jamais sur la limite de la matrice. Le **centre** de la forme se trouve en $(G_x/2,\; G_y/2)$, et les *demi-axes* sont $r_x = D/2$ et $r_y = D/2$ (ou $W/2$ et $H/2$ pour une ellipse).

3. L'équation fondamentale : cercle et ellipse

Toute la théorie du projet débute avec l'équation implicite de l'ellipse centrée à l'origine avec demi-axes r_x et r_y :

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

Lorsque $r_x = r_y = r$, l'ellipse dégénère en une **circonférence** de rayon r. Les points de l'*intérieur* satisfont ≤ 1 ; ceux de l'*extérieur*, > 1. C'est l'inégalité qui définit si un pixel appartient ou non à la forme. Les trois algorithmes sont, au fond, trois manières différentes d'appliquer (ou d'approcher) ce test sur une grille.

En déplaçant le centre vers (c_x, c_y) et en évaluant au centre de chaque cellule :

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

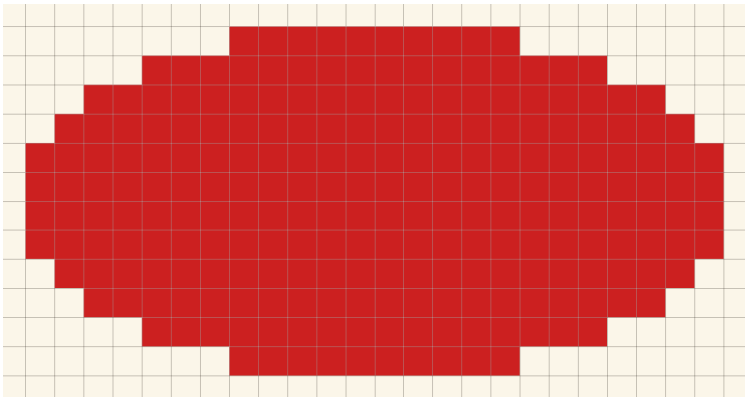

Fig. 1 : Ellipse de demi-axes $W = 24$ et $H = 12$ sur la grille discrète.

4. Algorithme euclidien (test de distance)

Idée. Pour chaque ligne j, trouver analytiquement les colonnes i qui se situent à l'intérieur de l'ellipse. En résolvant l'équation de l'ellipse en x, étant donné y :

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

Pour une ligne j dont le déplacement vertical est $d_y = j + \frac{1}{2} - c_y$, la *demi-largeur horizontale* de l'ellipse à cette hauteur est :

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

Si $d_y^2 / r_y^2 \geq 1$, la ligne ne coupe pas l'ellipse et est sautée. Sinon, toutes les colonnes i telles que $c_x - dxM \leq i + \frac{1}{2} \leq c_x + dxM$ sont remplies. Les bornes entières viennent de :

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

Justification. Le terme $+\frac{1}{2}$ provient de la convention selon laquelle la cellule d'indice i a son centre en $i + \frac{1}{2}$. Les fonctions $\lceil \cdot \rceil$ et $\lfloor \cdot \rfloor$ réalisent la conversion de l'intervalle continu vers des indices entiers, garantissant que seules les cellules dont le **centre** appartient à l'intérieur de l'ellipse soient incluses. Cette formulation produit l'approximation discrète la plus proche de la courbe continue et, par conséquent, la silhouette de plus grande douceur visuelle. Pour de petits diamètres, cependant, le résultat peut présenter un caractère pixel art moins marqué que celui des autres algorithmes.

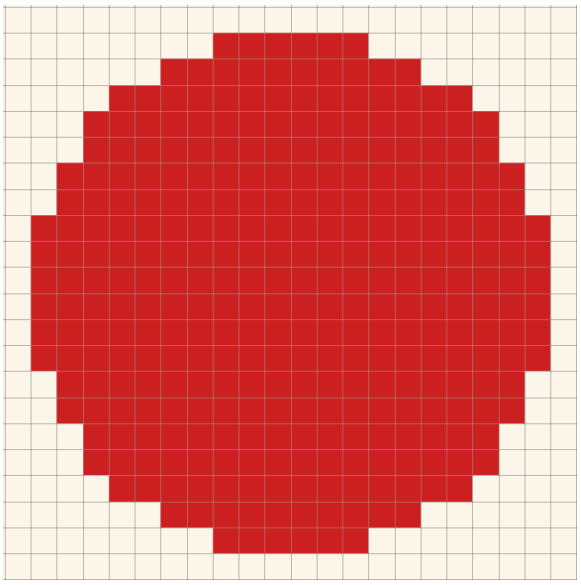

Fig. 2 : Algorithme euclidien à $D = 20$.

5. Algorithme de Bresenham (point milieu entier)

L'algorithme de Bresenham, publié en 1965 pour le tracé de segments de droite, fut ensuite étendu aux cercles et généralisé par Pitteway et Kappel aux ellipses arbitraires. Sa propriété distinctive est de se passer d'opérations en virgule flottante et d'extraction de racine carrée : la détermination du pixel suivant n'utilise que des **additions et comparaisons sur entiers**, via une *fonction de décision* qui évalue de quel côté de la courbe analytique se trouve le point milieu entre les deux pixels candidats.

5.1 Cas circulaire

Pour un cercle de rayon entier R, on part de $(x{=}0,\, y{=}R)$ avec paramètre de décision initial :

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

À chaque pas (dans l'octant de 0 à 45°) :

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

Justification. La fonction d approche, à chaque itération, la valeur de l'équation implicite du cercle évaluée au **point milieu** entre les deux options de pixel candidat :

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

Le signe de d indique si le point milieu se trouve à l'intérieur du cercle (auquel cas on n'avance qu'horizontalement) ou à l'extérieur (auquel cas la coordonnée verticale est également décrémentée). Les huit octants du cercle sont obtenus par application des symétries du groupe diédral D_4 à l'octant calculé, ce qui correspond, dans le code, aux huit appels à `span(...)`. Le tracé résultant exhibe le motif en escalier caractéristique des approximations discrètes de courbes lisses.

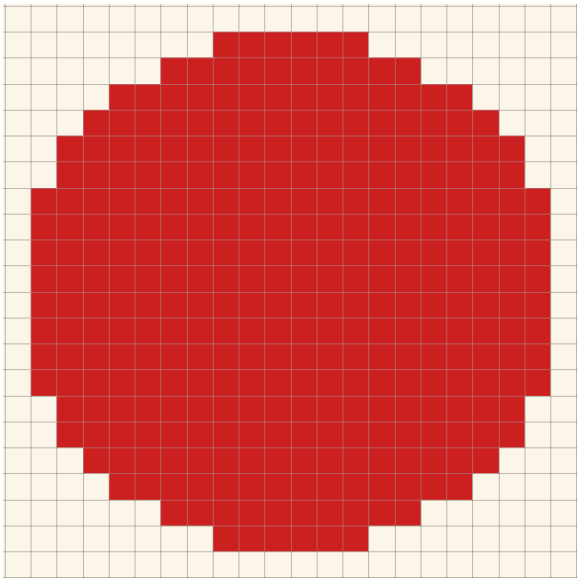

Fig. 3 : Algorithme de Bresenham à $D = 20$.

5.2 Cas elliptique (deux régions)

Pour une ellipse de demi-axes a, b, l'algorithme divise le premier quadrant en **deux régions**, délimitées par le point où la tangente forme un angle de 45° :

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

Les paramètres de décision initiaux sont :

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

À chaque itération, p est mis à jour par des incréments précalculés (tous entiers, après multiplication par 4). Le résultat est l'arithmétique minimale possible par pixel : *un test de signe et deux additions*. Le code utilise `Math.floor` en calculant a et b à partir de r_x et r_y pour éviter d'ajouter une ligne supplémentaire en dehors de la *bounding box* lorsque r_y n'est pas entier.

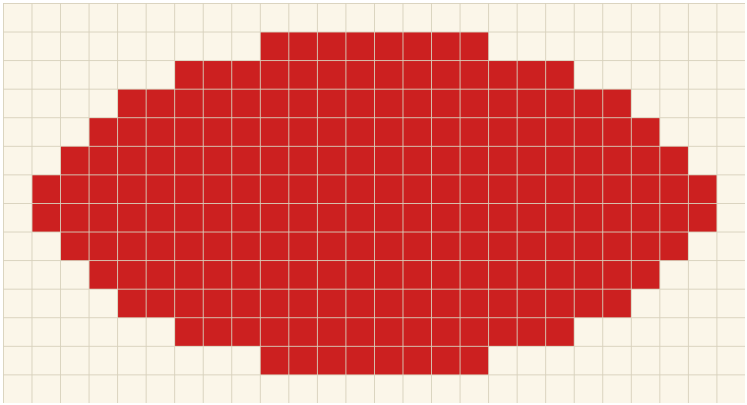

Fig. 4 : Algorithme de Bresenham appliqué à léllipse $W = 24$, $H = 12$.

6. Algorithme de seuil (Threshold / couverture de coin)

Cet algorithme demande : *existe-t-il un coin de la cellule à l'intérieur de l'ellipse ?* Pour chaque cellule (i, j), le code cherche le coin le plus proche du centre de la forme (projection du centre sur les arêtes de la cellule) :

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

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

La valeur 0,94 est un *seuil empirique* légèrement inférieur à 1,0. Son rôle est d'éliminer l'artefact dit *pic tangentiel* de 1 pixel, qui survient aux quatre points cardinaux (N, S, E, O), où la courbe est tangente à une arête entière de la cellule. Sans cette réduction, l'algorithme inclurait une cellule supplémentaire dont la contribution n'améliore pas la représentation visuelle de la forme.

Parmi les trois algorithmes, c'est celui qui produit la silhouette de plus grande aire pour un même diamètre D, puisque la condition d'inclusion est la moins restrictive : il suffit qu'un seul coin de la cellule satisfasse l'inégalité de l'ellipse pour que la cellule entière soit remplie.

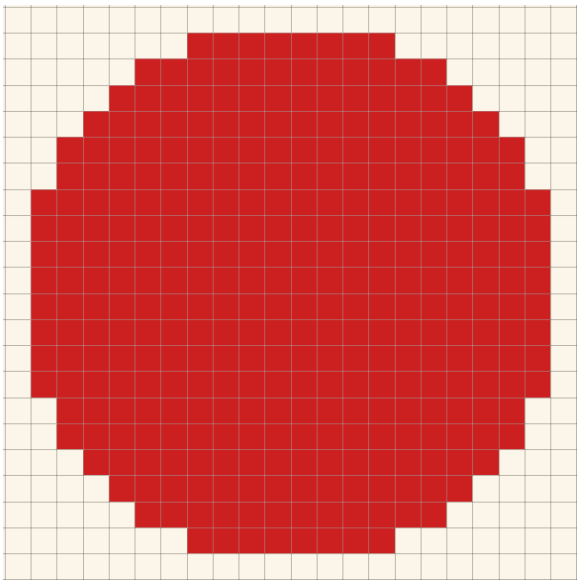

Fig. 5 : Algorithme à seuil à $D = 20$.

7. Modes de rendu : Plein, Fin, Épais

Disposant d'une matrice $f[j][i] = 1$ pour les cellules internes, le projet dérive trois styles visuels par **topologie discrète** :

7.1 Plein

Renvoie f sans modification. Tous les pixels internes.

7.2 Fin (outline)

Un pixel appartient au contour fin s'il est rempli *et* possède au moins un voisin orthogonal (N, S, E, O) *hors* de la forme. En notation logique :

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

Géométriquement : c'est le **bord discret** de la forme, l'analogue discret de la frontière topologique $\partial F = F \cap \overline{F^{\mathrm{c}}}$.

7.3 Épais (thick)

Le mode épais ajoute aux pixels de bord les **ponts diagonaux** : cellules internes qui possèdent simultanément un voisin-bord horizontal *et* un voisin-bord vertical. Cela ferme les diagonales d'un pixel que le mode fin laisse ouvertes, utile lorsque

le contour doit paraître stable dans une scène (sans trous à 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. Calcul d'aire discrète

La fonction `area2D` **compte** simplement les cellules marquées dans f. C'est la *mesure de comptage*, qui approche l'intégrale de Riemann de la fonction indicatrice de l'ellipse :

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

Pour D grand, aire discrète/aire continue $\to 1$. Pour D petit, la différence entre algorithmes est visible : Threshold tend à surestimer ; Euclidien reste proche de l'aire idéale ; Bresenham se situe quelque part entre les deux.

9. De la 2D à la 3D : l'équation de l'ellipsoïde

En 3D, la forme fondamentale est l'**ellipsoïde** centré en (c_x, c_y, c_z) avec demi-axes r_x, r_y, r_z. Son équation est la généralisation naturelle de l'ellipse :

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

Lorsque $r_x = r_y = r_z$, on retrouve la **sphère**. En évaluant au centre de chaque voxel :

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

10. Voxelisation et calcul de volume

Le volume continu de l'ellipsoïde est un résultat classique du calcul intégral, obtenu par intégration sur des disques :

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

La version discrète parcourt chaque voxel de la boîte $D_x \times D_y \times D_z$ et incrémente un compteur lorsque $a_x^2 + a_y^2 + a_z^2 \leq 1$. C'est l'algorithme `voxelVolume`.

Coque (shell) vs. intérieur. Pour le mode *filled*, le projet n'affiche que les voxels possédant au moins un 6-voisin hors de l'ensemble conservé (c'est-à-dire les voxels visibles depuis la surface externe ou depuis la face de coupe). Pour *thin*, seule la **surface** de l'ellipsoïde complet est affichée (1 voxel d'épaisseur).

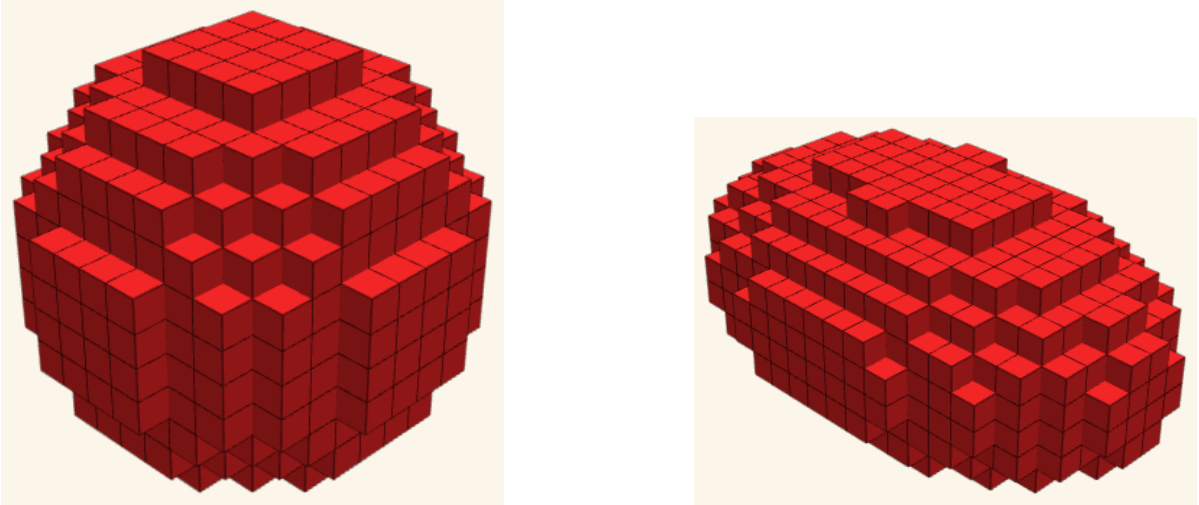

Fig. 6 : Sphère ($D = 12$) et ellipsoïde ($W = 20$, $H = 10$, $D = 12$) voxelisés.

11. Coupes géométriques : plans et la coupe diagonale à 45°

Pixel Round permet de couper le solide par **trois plans**. Chaque coupe est une *inégalité linéaire* qui élimine des voxels :

$$\begin{array}{lll} \text{Axe X :} & x \geq cut & \Rightarrow \text{éliminé} \\ \text{Axe Y :} & y \geq cut & \Rightarrow \text{éliminé} \\ \text{Diagonale :} & x + y \geq cut & \Rightarrow \text{éliminé} \end{array}$$

Les plans X et Y sont perpendiculaires aux axes de coordonnées, de vecteurs normaux $(1, 0, 0)$ et $(0, 1, 0)$. Le *plan diagonal* a pour vecteur normal $(1, 1, 0)/\sqrt{2}$ et coupe à 45° dans le plan XY : mathématiquement, il s'agit de la famille de plans $x + y = k$. Comme le maximum de $x + y$ dans une boîte $D_x \times D_y$ vaut $D_x + D_y$, le curseur de coupe diagonale a une amplitude $D_x + D_y$, tandis que X et Y ont des amplitudes D_x et D_y, respectivement. La coupe est **proportionnelle** : le code conserve un pourcentage *cutPct* et recalcule la limite par

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

de sorte que 50% sur un axe reste 50% lorsque l'utilisateur change d'axe ou redimensionne.

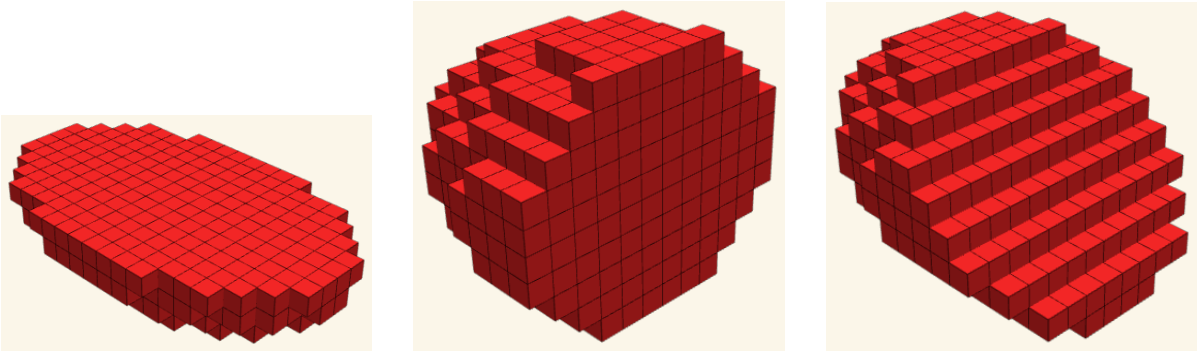

Fig. 7 : Coupes planes à 50% sur les axes Y, X et diagonale (de gauche à droite).

12. Mode épais 3D : tridimensionnalisation par tranches

Pour le mode *thick* en 3D, le projet applique l'algorithme épais 2D à **toutes les tranches** suivant les trois axes : XY pour chaque z, XZ pour chaque y, YZ pour chaque x. Il effectue ensuite l'*union* des résultats. La motivation géométrique est l'étanchéité : l'ensemble minimal de voxels qui bouchent toutes les diagonales sans doubler l'épaisseur de la coque. Formellement, en notant $S_z(z)$ la tranche XY à la hauteur z et T l'opérateur épais 2D :

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

Le résultat est ensuite intersecté avec l'ensemble préservé par la coupe. La théorie sous-jacente est la **topologie numérique** : l'opérateur épais garantit la *26-connexité* de la coque (voisins incluant les diagonales), utile pour construire dans Minecraft où les voxels diagonaux ne se touchent pas par face.

13. Caméra 3D : coordonnées sphériques

La caméra 3D orbite l'objet à une distance r avec deux angles — θ (azimut, dans le plan horizontal) et φ (polaire, depuis l'axe vertical). C'est la **convention physique** des coordonnées sphériques, et la conversion vers les cartésiennes (la forme que comprend *three.js*) est :

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

Position initiale : $\theta = \pi/4$ (45°, perspective isométrique) et $\varphi = \pi/3$ (60° depuis le pôle Nord, légèrement au-dessus de l'équateur). Le glisser de la souris incrémente θ et φ directement ; la molette/pincée incrémente r. La simplicité de cette paramétrisation permet la rotation libre sans quaternions ni matrices explicites.

14. Auto-zoom : sphère englobante et FOV

Lorsque l'utilisateur modifie la taille de la figure ou réinitialise la caméra, le projet recalcule la **distance idéale** de la caméra pour que l'objet tienne dans la vue sous n'importe quel angle. L'idée est d'enfermer l'objet dans une *sphère de rayon* R (la moitié de la diagonale de l'AABB, axis-aligned bounding box) :

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

Une sphère possède une **projection invariante par rotation**, donc si elle tient dans la vue sous une orientation, elle tient sous toutes. Étant donné le FOV vertical de la caméra ($fov_V = 35°$ en radians), la distance nécessaire pour cadrer une sphère de rayon R est :

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

Le FOV horizontal découle du rapport d'aspect du canevas ($aspect =$ largeur/hauteur) par :

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

La distance finale est le maximum de $dist_V$ et $dist_H$, multiplié par 1,20 (20% de marge visuelle). Lorsque la figure est coupée, le code réduit D_x ou D_y à la taille restante, afin qu'une sphère coupée à 75% n'apparaisse pas minuscule dans un coin.

15. Teintes (shading) et arithmétique de couleur RVB

Les trois styles 3D (*Classic*, *Smooth*, *Blocks*) se distinguent par des **facteurs multiplicatifs** appliqués aux trois faces visibles de chaque voxel (dessus, côté éclairé, côté sombre). La fonction `shadeHex(hex, factor)` analyse l'hexadécimal en (R, G, B) et multiplie chaque canal, avec *clamp* dans $[0, 255]$:

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

C'est une *approximation linéaire* de la fonction d'éclairage Lambertienne idéale,

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

lorsque les trois facteurs correspondent au cosinus de l'angle entre la normale de chaque face et une direction de lumière fixe. En *Smooth*, les facteurs sont proches (faces similaires) ; en *Classic*, les facteurs diffèrent davantage (fort contraste). *Blocks* introduit en outre un renfoncement géométrique dans les voxels — une translation constante de chaque face vers l'intérieur.

16. Audio : fréquences et la gamme musicale

Les sons de l'interface sont synthétisés en temps réel via WebAudio. Chaque son est une **onde sinusoïdale pure** :

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

Les valeurs de f (en Hz) choisies par le projet ne sont pas arbitraires — elles approchent des notes musicales du *tempérament égal à 12 tons*, où chaque demi-ton multiplie la fréquence par $2^{1/12} \approx 1{,}0595$:

Son	Fréquence (Hz)	Note la plus proche	Usage
click	620	≈ D#5 / E♭5	boutons
hover	1100	≈ C#6	survol bureau
tick	1300	≈ E6	pas du curseur
ok	520 → 720 → 920	arpège ascendant	réinit. / téléchargement
pop	680 + 980	intervalle de quinte juste	bascules
error	200	≈ G#3	action invalide

Quinte juste (rapport 3 : 2) : le son *pop* mélange 680 Hz et 980 Hz ; 980/680 ≈ 1,44 (une quinte juste pure serait 1,5), suffisant pour sonner consonant. L'**arpège du son *ok*** présente des rapports successifs proches de tierces : 520 → 720 (rapport 1,38, proche de la tierce mineure tempérée de 1,2 légèrement élargie) et 720 → 920, configuration qui établit une cadence ascendante perçue comme conclusive. L'enveloppe a une durée comprise entre 12 et 160 ms et un gain maximal de 0,035 (environ −29 dBFS), niveau délibérément bas pour éviter la fatigue auditive en usage prolongé.

17. Conclusion

Le présent document a décrit, de manière systématique, les fondements mathématiques employés dans Pixel Round. En environ mille lignes de code JavaScript, le projet intègre des contributions de plusieurs domaines :

- **Géométrie analytique** : équations implicites de l'ellipse et de l'ellipsoïde comme critère primaire d'appartenance.
- **Algorithmes classiques de rastérisation** : Bresenham dans la formulation au point milieu, avec l'extension de Pitteway/Kappel pour les ellipses.
- **Topologie numérique** : opérateurs de bord discret, concepts de 4-, 6- et 26-connexité et composition par tranches.
- **Calcul intégral** : volume de l'ellipsoïde comme intégrale triple et son équivalent discret (mesure de comptage sur voxels).
- **Algèbre linéaire** : plans de coupe définis par des inégalités linéaires et projection perspective tridimensionnelle.
- **Trigonométrie** : paramétrisation de la caméra en coordonnées sphériques et ajustement automatique de la distance en fonction du FOV.
- **Synthèse de signaux** : ondes sinusoïdales pures et approximation du tempérament égal à douze tons.

L'observation centrale que l'étude permet de formuler est la suivante : sur une grille discrète, **il n'existe pas de représentation correcte unique** d'une courbe continue. Chaque algorithme présenté dans ce document correspond à une définition mathématique distincte du critère d'inclusion de cellules, et chaque définition produit une silhouette aux propriétés formelles propres — douceur dans l'algorithme euclidien, motif en escalier de Bresenham, couverture plus large dans le critère de couverture de coin. Le choix de l'algorithme est donc une décision technique qui doit tenir compte de l'objectif de représentation visuelle visé et des caractéristiques métriques souhaitées pour la forme résultante.