Université Gustave Eiffel — INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

Master Informatique 1

TD Infographie n°3

Modélisation

Construction de formes canoniques simples à partir de leur représentation paramétriques.

Les procédures d'affichage sont détaillées dans un autre support (cf. TD.2-Dessin).

Pour les formes géométriques simples, on construira des tableaux de vertex et normales directement à partir de l'équation paramétrique de la surface voulue (cf TD.0-Géométrie).

Nous allons ici mettre en place, progressivement, une modélisation un peu spéciale permettant un affichage en résolution adaptative. Mais dans un premier temps nous allons voir comment construire ces tableaux pour une résolution fixe.

① **Un premier exemple complet : le carré de côté 2, centré sur l'origine Ω, dans le plan** ($z = 0$)

Ce premier objet (qui resservira pour la modélisation du cube) est très simple mais permet de mettre en évidence certains petits problèmes, commun à tous les objets.

- la résolution :
  C'est simplement les nombres de subdivisions que l'on souhaite dans les directions $\vec{x}$ et $\vec{y}$. On choisira, pour la démonstation, des résolutions différentes n_x et n_y
  ☞ le nombre total de vertex et de normales que l'on va construire sera donc $(n_y * n_x)$.
  ☞ de là découlent les *pas* sur les deux directions paramétriques x et y : $\delta_x = \frac{2.}{(n_x-1)}$ et $\delta_y = \frac{2.}{(n_y-1)}$
  ☞ les coordonnées des $(n_y * n_x)$ vertex et normales sont alors simplement :

$$\forall (i,j) \in [0,n_y] \times [0,n_x], \quad P[i*n_x+j] = \begin{vmatrix} -1. + j*\delta_x \\ -1. + i*\delta_y \\ 0. \end{vmatrix} \quad \text{et} \quad \vec{N}[i*n_x+j] = \begin{vmatrix} 0. \\ 0. \\ 1. \end{vmatrix}$$

- l'affichage :
  Cette surface étant simple et très régulière, un rendu en `GL_QUADS` (cf. TD.2-Dessin) sera largement suffisant et facile à mettre en oeuvre.

```
-----------------------------------------------------
  glBegin(GL_QUADS);
    for (i=0;i<ny-1;i++) /* on s'arrête à l'avant-dernier sur y ... */
      for (j=0;j<nx-1;j++) /* on s'arrête à l'avant-dernier sur x... */
      { /* les 4 vertex, dans le bon ordre - toute permutation circulaire est valable */
        k = (i  )*nx+(j  ); g3x_Normal3d(P[k]); g3x_Vertex3d(P[k]);
        k = (i  )*nx+(j+1); g3x_Normal3d(P[k]); g3x_Vertex3d(P[k]);
        k = (i+1)*nx+(j+1); g3x_Normal3d(P[k]); g3x_Vertex3d(P[k]);
        k = (i+1)*nx+(j  ); g3x_Normal3d(P[k]); g3x_Vertex3d(P[k]);
      }
  glEnd();
-----------------------------------------------------
```

- les raccordements :
  Le calcul des pas introduisant des erreurs dues au format des réels IEEE, selon les valeurs de n_x et n_y, les dernières valeurs échantillonnées n'ont que peu de chance d'être exactes : $(-1+(n_x-1)*\frac{2.}{n_x-1}) \simeq +1.$

  Cette légère erreur pourra s'avérer clairement visible à l'affichage (raccordement des faces du cube, par exemple).

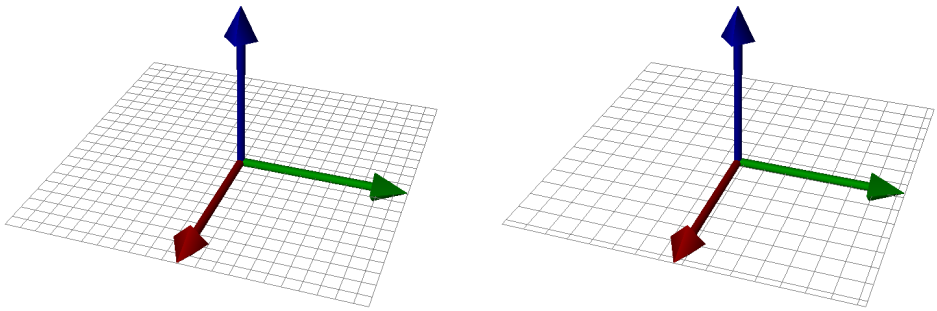
  Il faut alors prévoir une "rustine" dans la boucle de construction des vertex. Il suffit de forcer la position du dernier échantillon sur la valeur voulue.

Concrètement, la boucle de construction des vertex, pour la face carrée serait :

```
-----------------------------------------------------
double y,x,dy=2./ny,dx=2./nx;
for (i=0;i<ny-1;i++) /* on s'arrête à l'avant-dernier sur y ...     */
{
  for (j=0;j<nx-1;j++)    /* on s'arrête à l'avant-dernier sur x... */
    P[i*nx+j]=(G3Xpoint){(-1.+j*dx),(-1.+i*dy),0.};
  j=nx-1;                 /* ... et on raccorde sur x=+1            */
  P[start+i*MAXE+j]=(G3Xpoint ){+1,(-1.+i*dy),0.};
}
i=ny-1;              /* ... et on raccorde sur y=+1                */
for (j=0;j<nx-1;j++)      /* on s'arrête à l'avant-dernier sur x... */
  P[i*nx+j]=(G3Xpoint){(-1.+j*dx),+1,0.};
j=nx-1;                   /* ... et on raccorde sur (x=+1,y=+1)     */
P[i*nx+j]=(G3Xpoint ){+1,+1,0.};
-----------------------------------------------------
```

Cette "rustine", nécessaire pour tous les échantillonnages, allonge l'écriture du code (pas son exécution) mais garantie un raccordement parfait des faces.

☞ Un problème similaire apparaîtra lors des parcours adaptatifs des tableaux (cf. ③). Il nécessitera des adaptations de la boucle d'affichage.

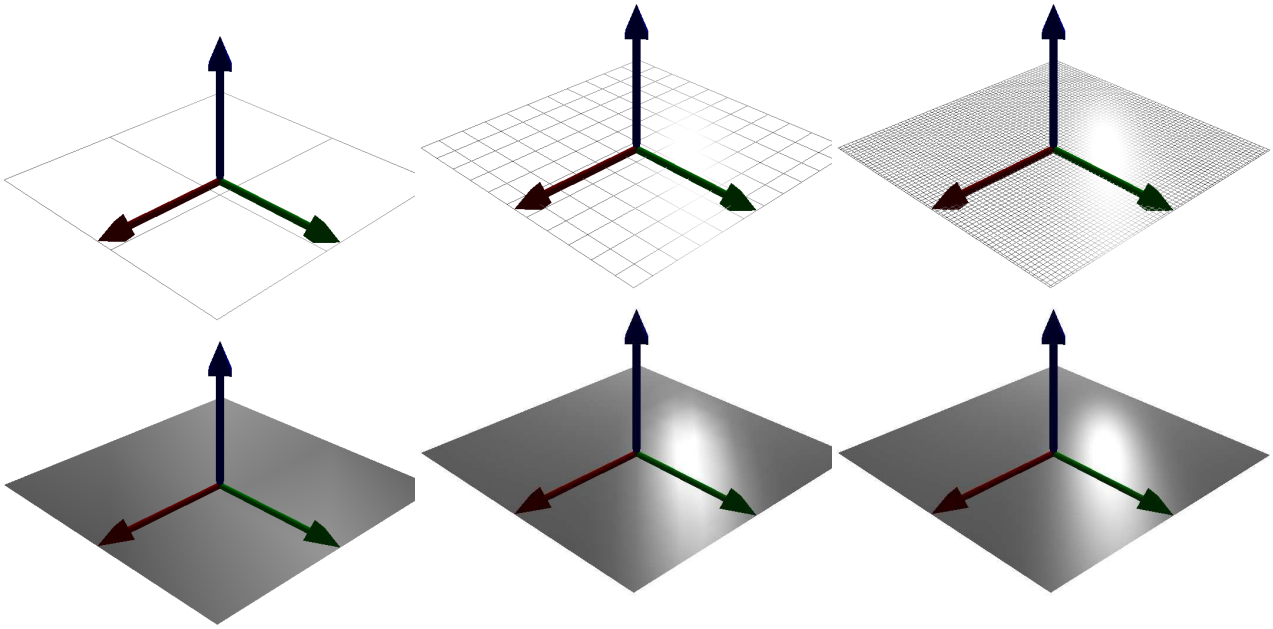


2 exemples avec des résolutions différentes : à droite, le besoin d'un "raccordement" est flagrant

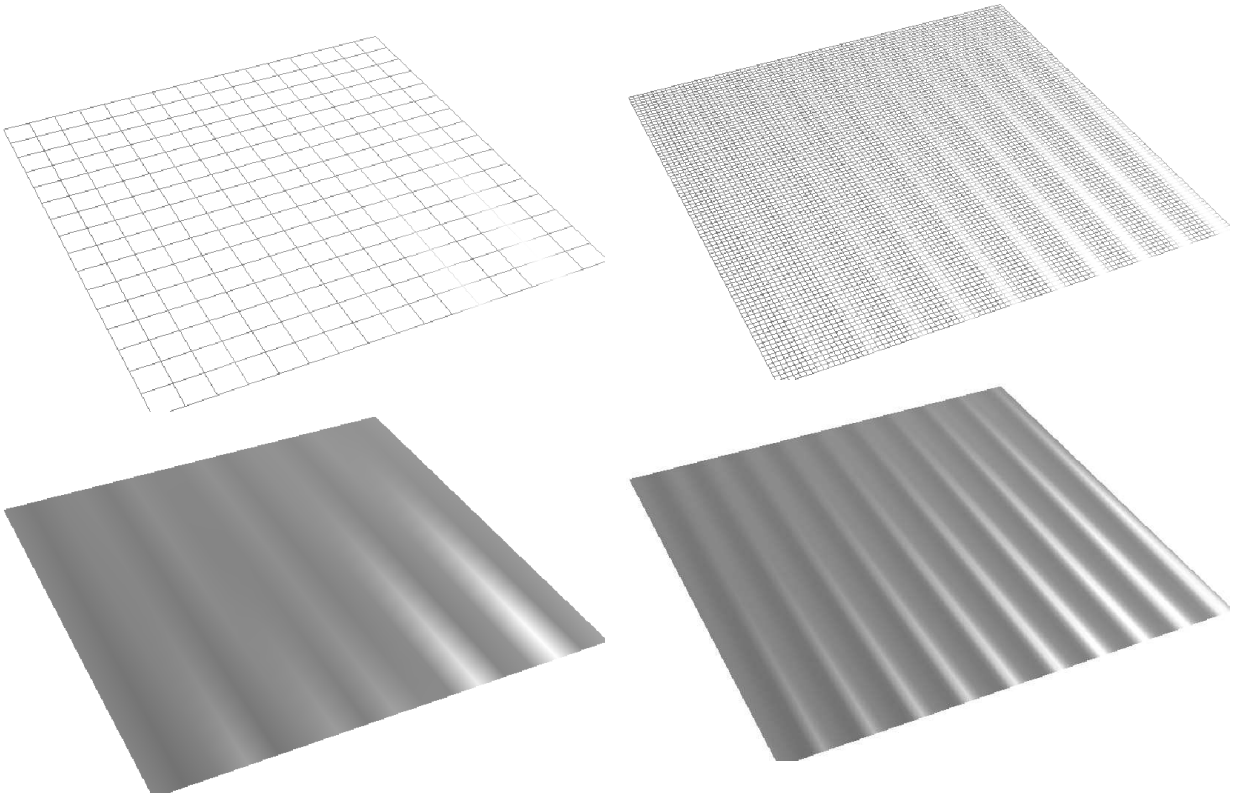
- <u>l'intérêt de cette subdivision</u> :

  Dans le cas d'une surface plane telle que la face carrée, on peut raisonnablement se poser la question de son utilité. En effet la définition des 4 sommets du carré suffit à définir la surface, et les normales sont identiques en tout point.

  L'intêret apparait lorsqu'on modélise les interactions avec la lumière (cf. TD.4-Illum) : avec une faible résolution, l'interaction lumière/matière est "globale" (pas de détails) alors qu'avec une forte résolution, la lumière interagit différemment avec les diverses facettes.



Il devient aussi possible de mettre en oeuvre très simplement des "effets", tel que le Normal-Mapping comme dans l'exemple ci-dessous, pour un effet "tôle ondulée" (simple pertubation des normales par des fonctions sin/cos). On voit que l'effet dépend directement de la résolution.



② Un second exemple complet : la Sphère Canonique $\mathcal{S}_{(\Omega,1.)}$

Les équations paramétriques complètes (sommets et normales) étant connues (cf TD.0-Géométrie), il reste à mettre en oeuvre leur discrétisation.

- la résolution :

  la sphère paramétrique est définie par analogie au globe terrestre, avec 2 paramètres angulaires $\theta \in [0, 2\pi]$ représentant la longitude (angle $\widehat{(x \to y)}$) et $\phi \in [0, \pi]$ représentant la latitude (initialisée au "pôle nord" $(0, 0, +1.)$, dirigée vers le "pôle sud" $(0, 0, -1.)$).

  ☞ la résolution est alors la donnée des nombres de "méridiens" n_m et de "parallèles" n_p.

  ☞ le nombre total de vertex et de normales que l'on va construire sera donc $(n_m * n_p)$.

  ☞ de là découlent les *pas angulaires* sur les deux directions paramétriques θ et ϕ :

$$\theta_m = \frac{2\pi}{n_m} \quad \text{et} \quad \phi_p = \frac{\pi}{n_p}$$

  ☞ les coordonnées des $(n_p * n_m)$ vertex sont alors simplement :

$$\forall (i, j) \in [0, n_p[\times[0, n_m[, \quad P[i * n_m + j] = \begin{vmatrix} \sin(i * \phi_p) * \cos(j * \theta_m) \\ \sin(i * \phi_p) * \sin(j * \theta_m) \\ \cos(i * \phi_p) \end{vmatrix}$$

  ☞ et dans ce cas (TRES particulier) les normales sont strictement identiques aux vertex (`memcpy`).

- les raccordements :

  Dans le cas de la sphère, la question des raccordements se pose de manière un peu différente. En effet, contrairement au carré précédent c'est une surface doublement fermée : les méridiens se raccordent pour former des cercles et les parallèles convergent vers des points uniques aux pôles nord et sud.

  Le cas des pôles pose plusieurs problèmes. Mathématiquement, la formulation paramétrique précédente fait que l'on crée n_m points superposés au pôle nord $(0, 0, +1)$ et pareil au pôle sud $(0, 0, -1)$. Pour limiter les effets indésirables liés à l'approximation sur les réels (apparition de 'trous') il est préférable de les traiter à part.

  Concrètement, la boucle de construction des vertex, pour la sphère serait :

```
-------------------------------------------------------
int i,j;
double theta =(2*PI)/Nm;
double phi   =(  PI)/Np;
double r,x,y,z;
/* le pôle nord : Nm point superposés (forcés) */
i=0;
for (j=0;j<Nm;j++) P[i*Nm+j]=(G3Xpoint){0.,0.,+1.};
/* pour chaque parallèle (hors pôles -- cf bornes de la boucle) */
for (i=1;i<Np-1;i++)
{
  z = cos(i*phi);
  r = sin(i*phi);
  for (j=0;j<Nm;j++)
    P[i*Nm+j]=(G3Xpoint){ r*cos(j*theta), r*sin(j*theta), z};
}
/* le pôle sud : Nm point superposés (forcés) */
i=Np-1;
for (j=0;j<Nm;j++) P[i*Nm+j]=(G3Xpoint){0.,0.,-1.};
-------------------------------------------------------
```

  remarque : un esprit avisé pourrait objecter qu'il est ridicule de créer autant de points superposés puisque deux suffisent (un à chaque pôle).
  ☞ c'est exact, mais la suite (constitution des facettes correctement orientées) deviendrait beaucoup plus ardue ! Libre à vous d'essayer....

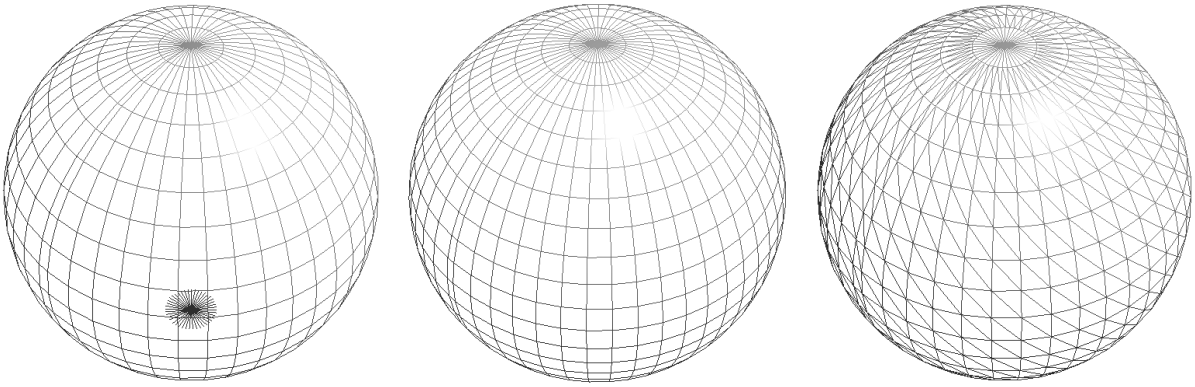
- la constitution des faces :

  Dans le cas de la sphère il est tentant de procéder avec des `GL_QUADS` puisque les données (vertex) sont directement caractérisées par deux paramètres. C'est d'ailleurs comme cela que procède <glut> pour créer sa `glutSolidSphere`.

  ☞ néanmoins, là encore les pôles posent problème : les `GL_QUADS` que l'on y forme ont 2 sommets superposés ce qui rend OpenGl inapte à interpoller 2 facettes triangulaires (il produit 2 facettes sans orientation définie – cf. dans l'image ci-dessous, à gauche, le pôle sud apparaît alors qu'il ne devrait pas).

  ☞ il est donc préférable de traiter les pôles à part en utilisant des `GL_TRIANGLES` (cf. au centre)

  ☞ On peut aussi tout afficher en mode `GL_TRIANGLES` (à droite).



représentation en `QUADS` (gauche), `QUADS+TRIANGLES` (centre) et `TRIANGLES` (droite)

③ Résolution adaptative – routines d'affichage

- principe général :

  L'idée est de ne créer qu'un seul exemplaire de chaque forme de base, caractérisée par ses deux tableaux de vertex et normales, en *résolution maximale* (ie. avec un très grand nombre de vertex), et de permettre un affichage à la résolution que l'on souhaite en fonction de la taille relative de l'*instance* de l'objet dans la scène.

  Par exemple, dans une scène contenant plusieurs sphères (éventuellement déformées) de tailles variables, on considèrera que :
  - la résolution maximale correspond à une sphère de rayon 1.
  - toutes les sphères présentent dans la scène sont de rayon $r \leq 1$.

  Alors, une sphère de rayon $r = 0.1$ sera affichée avec une résolution $\frac{1}{10}$ (par rapport à la résolution maximale) : on constitue les facettes à la volée (ie. à l'affichage) en 'piochant' un vertex sur 10, dans les 2 directions (longitude/latitude). On aura ainsi une sphère facétisée dont la face "de base" garde une taille à peu prés constante (on ne peut garantir une exactitude parfaite).

  De même, il faudra faire en sorte que, pour 2 objets différents (une sphère et un cube, par exemple) à la même échelle dans la scène, la taille de la facette de base soit visuellement à peu prés équivalente (là encore, impossible d'être exact).

  ☞ cf. pages suivantes pour le calcul de ces tailles optimales et des images d'exemples.

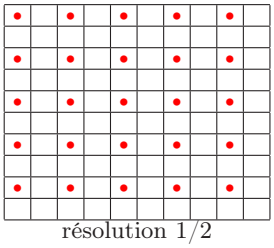
- raccordements adaptatifs (sur un cas concret) :

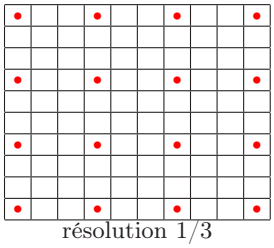
  Reprenons le cas du Carré $\mathcal{C}_{(\Omega,2,(z=0))}$ et supposons qu'il ait été défini (dans sa forme canonique) avec des résolutions $n_y = n_x = 10$ (on a donc au total 10^2 vertex).

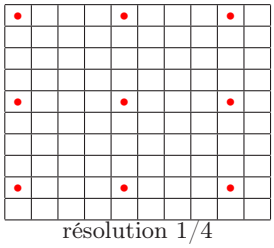
  $$\mid P_{(0,0)}...P_{(0,9)} \mid P_{(1,0)}...P_{(1,9)} \mid P_{(2,0)}...P_{(2,9)} \mid ..... \mid P_{(8,0)}...P_{(8,9)} \mid P_{(9,0)}...P_{(9,9)} \mid$$

  Pour afficher 3 instances de l'objet, aux résolutions $\frac{1}{2}$, $\frac{1}{3}$ et $\frac{1}{4}$, il faut parcourir le tableau des vertex en faisant des 'sauts' (colonnes et lignes) respectivement de 2, 3 et 4, et former des `GL_QUADS` avec chaque paquet de 4 vertex seléctionnés.

☞ On voit, graphiquement, que l'on atteint bien les bords opposés pour la résolution $\frac{1}{3}$, mais pas pour les 2 autres.

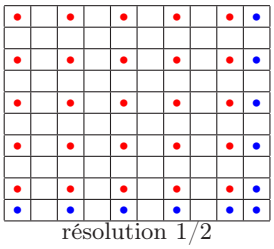

résolution 1/2

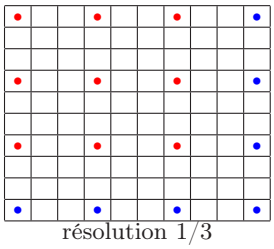

résolution 1/3

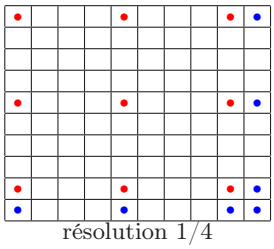

résolution 1/4

☞ il faut prévoir des "coutures" de raccordement.

Pour celà il faut simplement arrêter les boucles de parcours à l'avant dernière valeur (`for (i=0;i<ny-1;i+=pasy)` et `for (j=0;j<nx-1;j+=pasx)`) et terminer en raccordant directement sur la fin de la ligne et de la colonne (les points bleus●). Bien sûr les faces de "coutures" ne sont plus de la même taille que les autres.

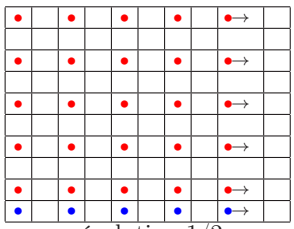

résolution 1/2

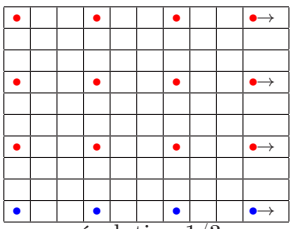

résolution 1/3

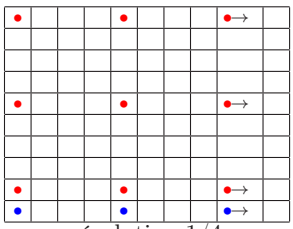

résolution 1/4

- <u>cas des surfaces "fermées"</u> :

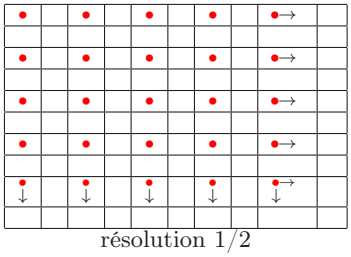
  Le cas du cylindre se déduit très simplement du cas de la face carré puisque les cercles du "bandeau" correspondent directement aux lignes des tableaux précédent qu'il suffit de boucler (●→) sur le début de la ligne.
  Verticalement, le traitement est le même que pour la facette carrée.

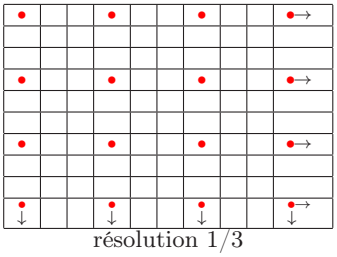

résolution 1/2

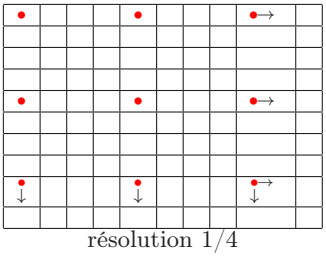

résolution 1/3


résolution 1/4

☞ le tore est finalement la surface la plus simple puisque le raccordement se fait par simple "bouclage" dans les deux directions (●→, ●↓) :


résolution 1/2


résolution 1/3


résolution 1/4

⑤ Adaption aux homothéties

Jusqu'à présent la gestion des résoltions était équivalente dans toutes les directions. Seule l'échelle générale de l'objet était prise en compte (on considérait des sphères plus ou moins grosses, mais qui restaient des sphères).

Si on veut faire en sorte que la résolution s'adapte indépendemment (dans la mesure du possible) en fonction des déformations, il faut prendre en compte les homothéties appliquées à chaque instance de l'objet pour définir des résolutions (et donc des pas de parcours des tableaux) dans les 3 directions.

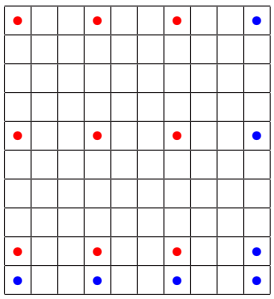
Rappel :
on considère ici que toutes les homothéties en jeu sont de rapport $\mathsf{h} < 1$, ce qui garantie que $((\mathsf{int})\,(\frac{1}{\mathsf{h}}) \geq 1.)$

• exemple : la facette carrée définie avec des résolutions $\mathsf{n_y} = \mathsf{n_x} = 10$

On lui applique une homothétie de rapports $(\mathsf{h_x} = 0.3,\ \mathsf{h_y} = 0.25)$ – la dimension z ne compte pas pour cet objet.

☞ On en déduit des 'pas' de parcours de $\mathsf{p_x} = \mathsf{Min}(1, (\mathsf{int})(1./\mathsf{h_x})) = 3$ selon x et $\mathsf{p_y} = \mathsf{Min}(1, (\mathsf{int})(1./\mathsf{h_y})) = 4$.

☞ Le parcours des tableaux donnerait (avec les raccordements aux bords) :



• le cube :
☞ les 3 dimensions sont indépendantes et chaque face se gère comme la facette carrée.
☞ On peut donc définir 3 "pas" indépendants directement déduit des valeurs d'homothétie.

• la sphère (surface fermée par des cercles) :
on note $\lfloor \mathsf{x} \rfloor$ la partie entière du réel x ($\lfloor \mathsf{x} \rfloor \leftrightarrow$`MAX(1,(int)(x))`)

☞ les dimension x et y sont liées (paramètre θ), la dimension z est indépendante.
☞ on peut définir un "pas moyen" pour θ : $\mathsf{p}_\theta = \lfloor 0.5 * (\frac{1}{\mathsf{h_x}} + \frac{1}{\mathsf{h_y}}) \rfloor$ ou mieux $\mathsf{p}_\theta = \left\lfloor \sqrt{(\frac{1}{\mathsf{h_x}})^2 + (\frac{1}{\mathsf{h_y}})^2} \right\rfloor$)
☞ pour ϕ : $\mathsf{p}_\phi = \lfloor \frac{\pi}{4 * \mathsf{h_z}} \rfloor$

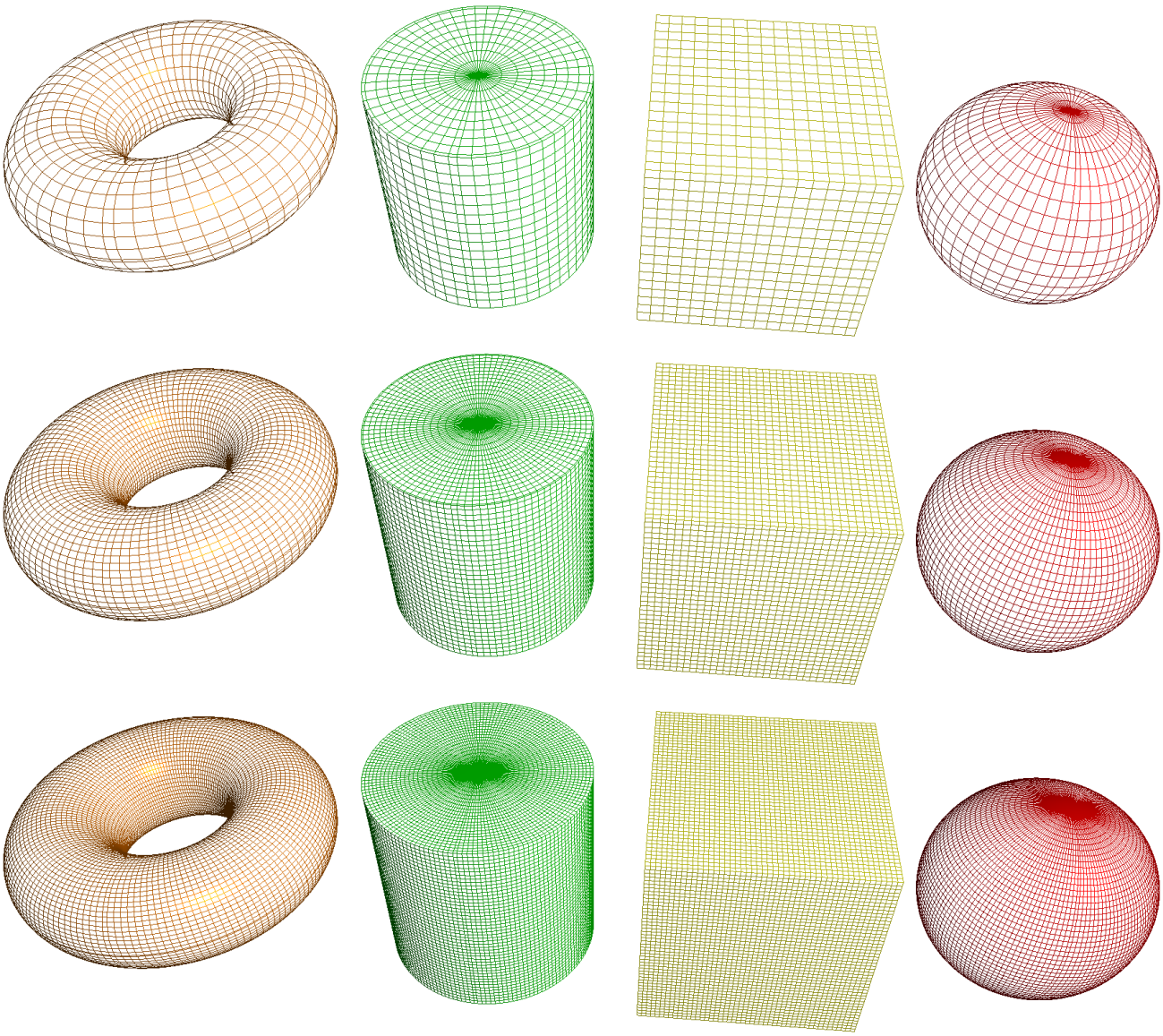
• le cylindre, le cône, le tore : mêmes techniques...

• résolution globale / taille de la facette "de base" ☞ dimension des tableaux canoniques :

on peut prendre comme référence la face carrée (de surface totale 4) de résolution maximale n (dans les 2 directions) ☞ la facette de base est de surface $\mathsf{s_0} = 4/\mathsf{n}^2$

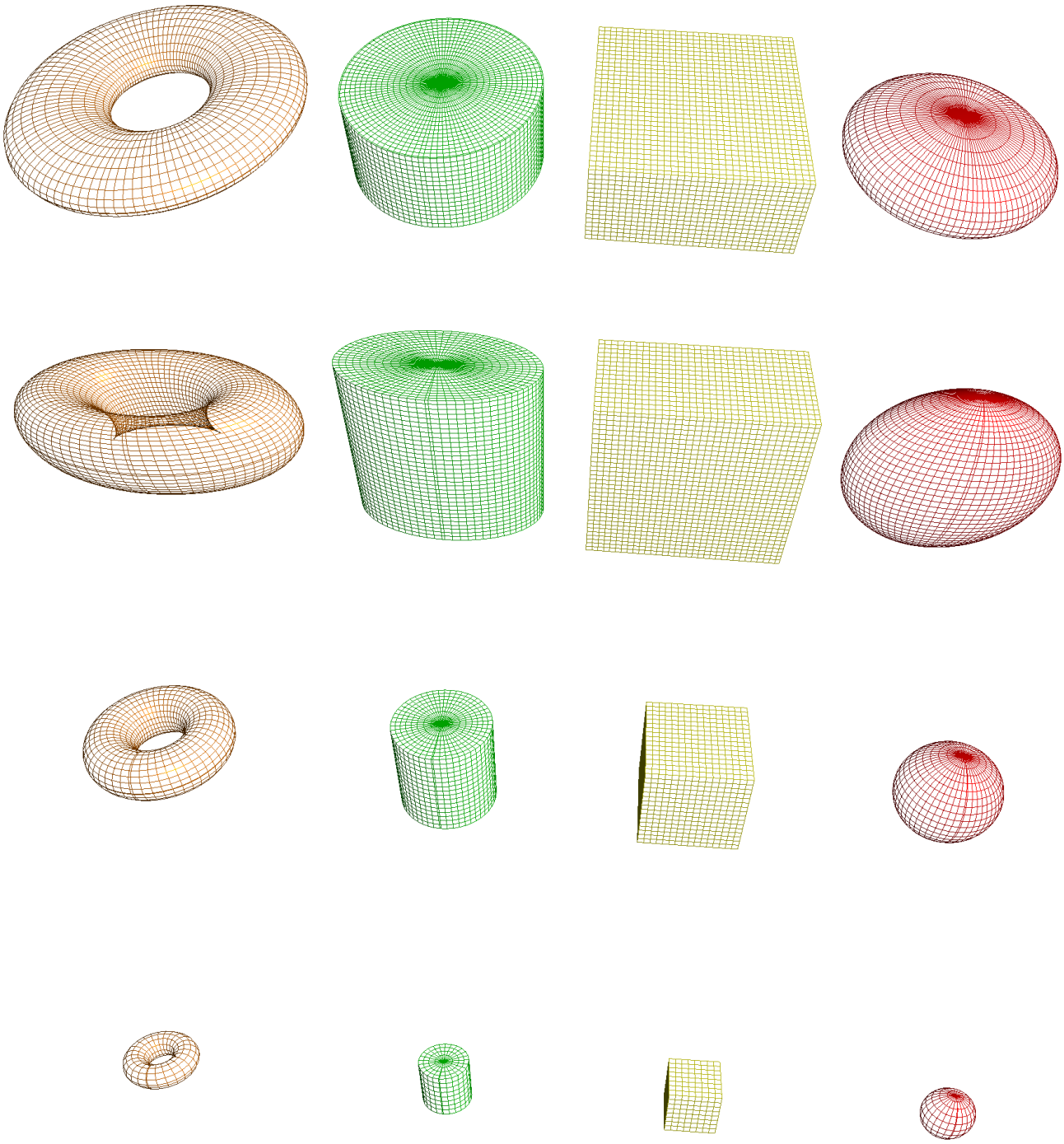
- cette face est donc définie par des tableaux de taille totale (n^2)
- le cube sera alors de taille $(6 * \mathsf{n}^2)$
- la sphère (de surface π) sera définie par des tailles de $(\lfloor \pi * \mathsf{n} \rfloor * \lfloor \pi * \mathsf{n}/2 \rfloor)$
- le cylindre sera défini par des tailles de $(\lfloor \pi * \mathsf{n} \rfloor * (\mathsf{n} + 2 * \lfloor \mathsf{n}/2 \rfloor))$ – les faces supérieure et inférieure
- le cône sera défini par des tailles de $(\lfloor \pi * \mathsf{n} \rfloor * (\mathsf{n} + \lfloor \mathsf{n}/2 \rfloor))$ – juste une face inférieure
- le tore sera défini par des tailles de $(\lfloor \pi * \mathsf{n} \rfloor * \lfloor (\pi * \mathsf{n})/2 \rfloor)$

Résolution adaptative indépendante de la forme



4 formes canoniques en basse, moyenne et haute résolution
☞ la taille de la "facette de base" reste à peu prés constante d'une forme à l'autre
☞ noter les raccordements sur toutes les formes

Résolution adaptative selon homothéties



Homothéties $(1.0, 1.0, 0.5)$, $(1.0, 0.5, 1.0)$, $(0.5, 0.5, 0.5)$ et $(0.25, 0.25, 0.25)$

☞ la taille de la "facette de base" reste à peu prés constante

☞ les raccordements s'adaptent en fonction des rapports d'échelle