Chapitre 3: Modélisation

INF5071 — Infographie

Alexandre Blondin Massé

Université du Québec à Montréal

Hiver 2019

Plan

Maillage

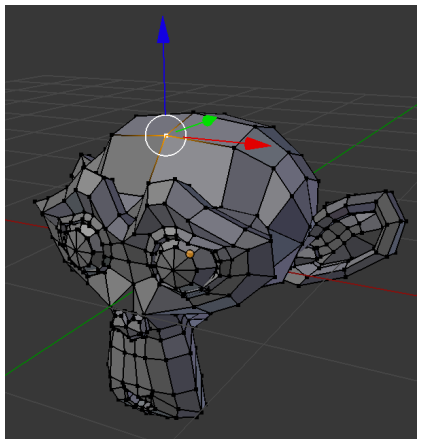
Définition

Un **maillage** est la donnée

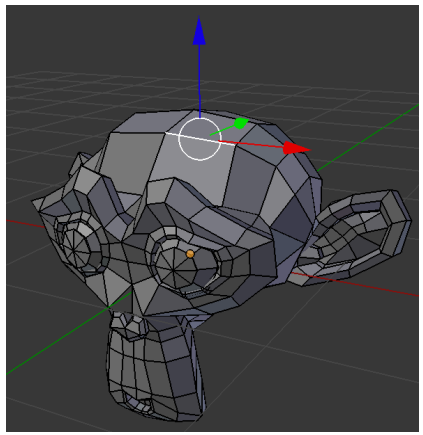
- d'un ensemble fini de **sommets** (*vertices*) $V \subset \mathbb{R}^3$
- d'un ensemble fini d'**arêtes** (*edges*) $E \subset \mathcal{P}_2(\mathbb{R}^3)$, où $\mathcal{P}_2$ est l'ensemble des **paires** de points dans $\mathbb{R}^3$
- d'un ensemble de **faces**

Remarque

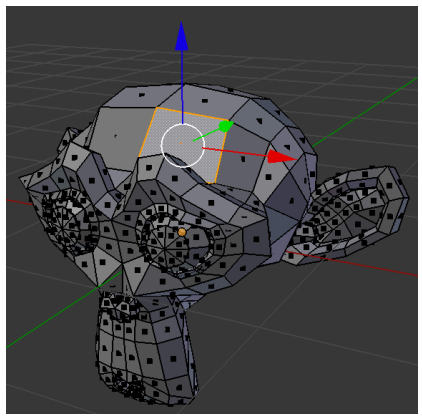
Un maillage permet de représenter une **surface** ou un **volume**



sommet



arête



face

Modèle

- Un maillage (*mesh*) est souvent accompagné d'**informations**
- On l'appelle alors un **modèle**

Exemple d'informations additionnelles

- Relations d'**adjacence** (*graphe*) entre les sommets/arêtes/faces
- **Orientation** des faces
- Coordonnées de **texture**
- Vecteurs **normaux**
- Transformations **implicites** (**non destructives**)
- **Matériaux**: carte de **diffusion** (*diffuse map*), carte de **normales** (*normal map*), de **relief** (*bump map*), de **luminosité** (*specular map*), d'**occlusion ambiante** (*ambient occlusion*)
- **Animations**: système articulé, déformations
- Propriétés **physiques**: corps rigide, object cinématique, etc.

Niveau de détails

- Souvent, un maillage est une **approximation** d'une surface mathématique
- Ou l'approximation d'un maillage **plus complexe**
- Comme les appareils **mobiles** supportent le 3D, il y a un regain d'intérêt pour les modèles 3D **de basse définition** (*low-poly*)

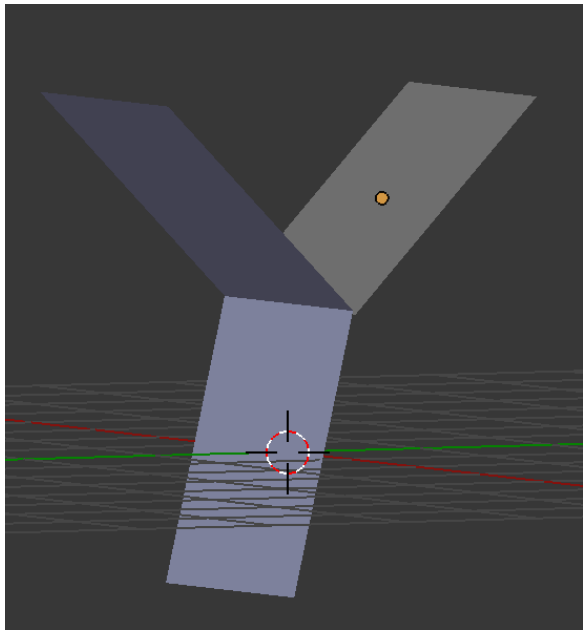
→ Définition **faible**: Moins de 1000 faces

→ Définition **moyenne**: de 1000 à 2500 faces

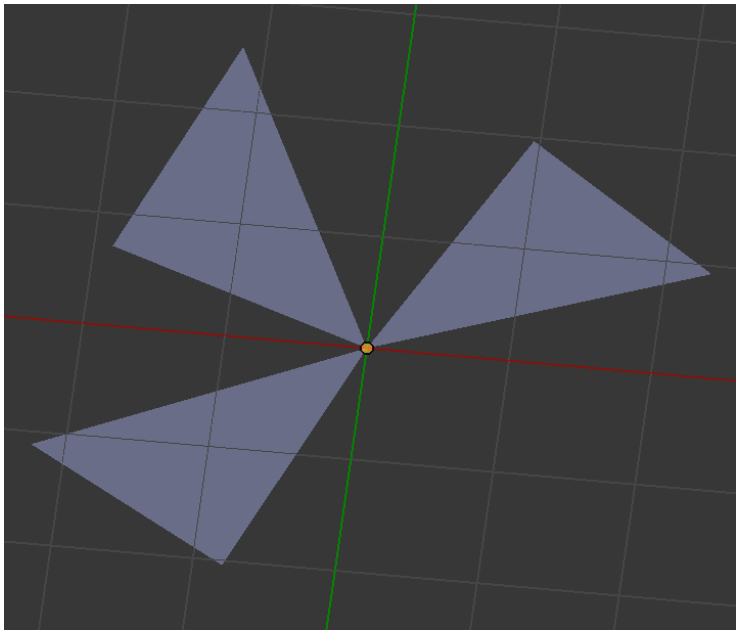
→ Définition **haute**: Plus de 2500 faces

- Dépend du **budget polygonal**: nombre de polygones pouvant apparaît dans une **scène**
- L'utilisation de **cartes** permet de réduire significativement le niveau de détails d'un modèle

Maillages dégénérés (1/2)



Arête incidente
à plus de deux faces



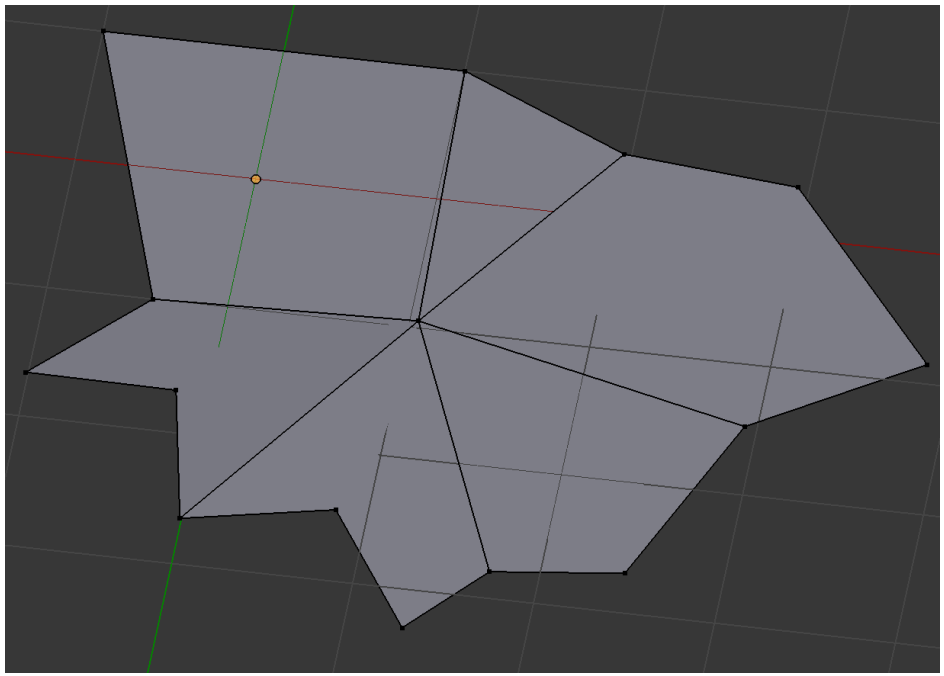
Faces voisines
non adjacentes

Maillages dégénérés (2/2)

- Afin de simplifier **plusieurs algorithmes** sur les maillages, on suppose que les propriétés suivantes sont vérifiées

$\rightarrow$ Chaque arête est **incidente** à **au plus** 2 faces

$\rightarrow$ Chaque sommet **induit** une chaîne simple



Coordonnées locales

- On applique des **transformations** aux maillages
- Il est donc pratique que **chaque maillage** ait son **propre espace vectoriel**
- Autrement dit, chaque maillage possède sa propre **origine** (Blender l'identifie par un point orange)
- Toutes les **transformations** se basent sur cette origine pour être appliquée

→ *Translation*: on translate l'origine

→ *Rotation*: autour d'un axe passant par l'origine

→ *Changement d'échelle*: par rapport à l'origine

Requêtes d'adjacence

La structure de données utilisée pour représenter un maillage **dépend** des opérations et des requêtes qu'on souhaite **supporter**

Les requêtes d'**adjacence** sont importantes:

- FV: tous les **sommets** d'une **face** donnée
- EV: les deux **sommets** d'une **arête** donnée
- VF: toutes les **faces** partageant un **sommet** donné
- EF: les **faces** partageant une **arête** donnée
- FE: toutes les **arêtes** autour d'une **face** donnée
- VE: toutes les **arêtes** autour d'un **sommet** donné

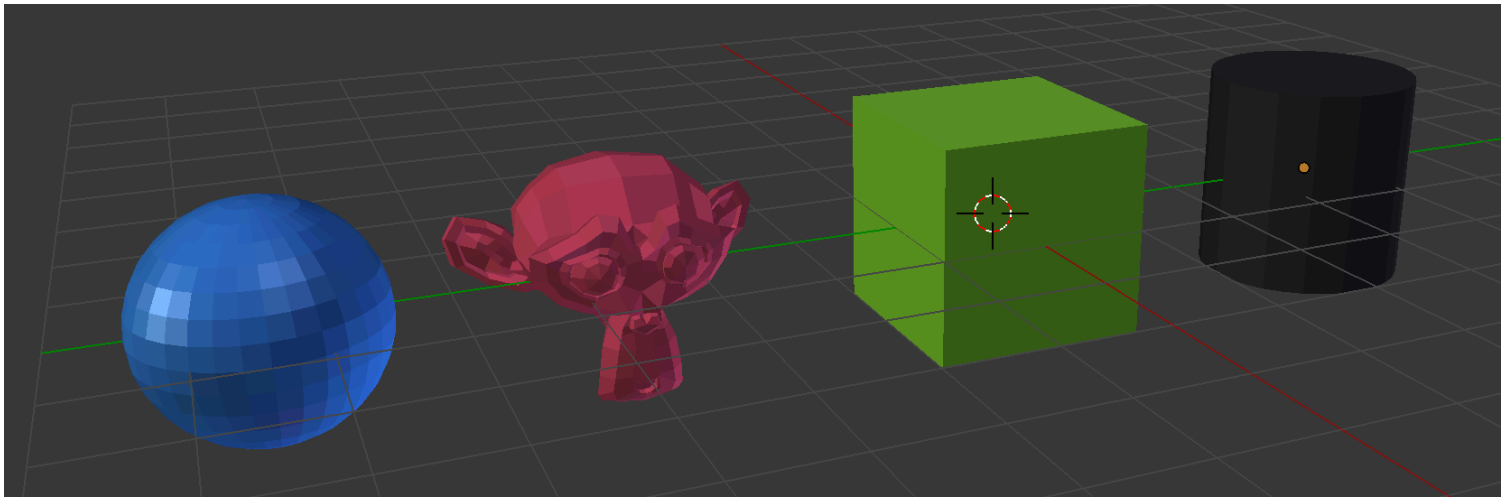
Représentation naïve

- Tout maillage peut être **triangulé**
- On représente les sommets par une **liste** de triplets (x_i, y_i, z_i), pour $i = 1, 2, \ldots, n$
- Puis les faces sont dans une **liste** de triplets (j_i, j_k, j_ℓ) décrivant les **indices** des sommets apparaissant dans **chaque face**
- C'est la **structure de base** d'OpenGL pour représenter un **maillage**
- **Efficace** pour les requêtes FV, EV et FE, utilisées dans les **rendus**
- **Inefficace** pour une requête EF qui s'effectue en $\mathcal{O}(|F|)$, où F est l'ensemble des **faces**

Matériaux

Matériaux de base

- L'application de **matériaux** à un maillage est un processus complexe
- Toutefois, la plupart des logiciels fournissent des **matériaux simples** par défaut.

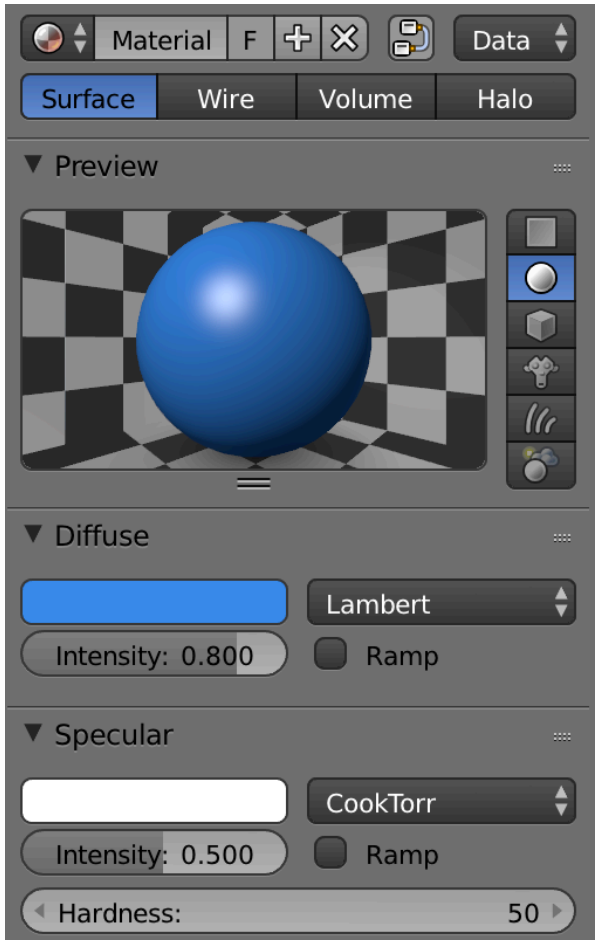


Diffuse/Specular

Diffuse: correspond à la **« couleur »** ou la « texture » de l'objet avec un éclairage **neutre**

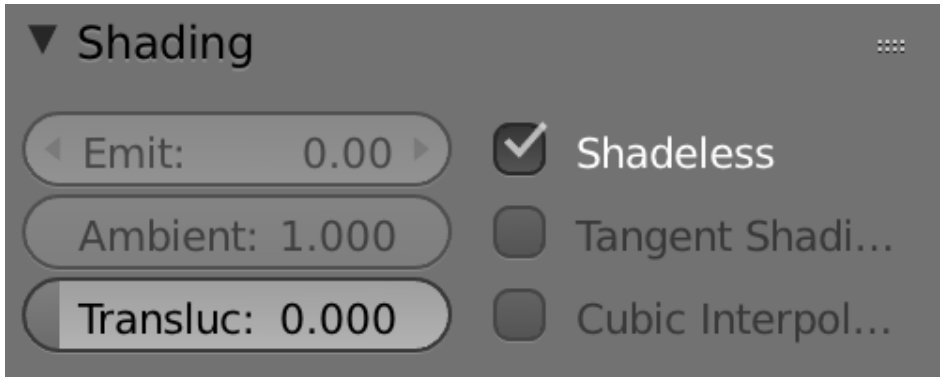
Specular: indique à quel point l'objet **réfléchit** la lumière:

- Un objet **réfléchissant** a un indice de spécularité près de 1.0
- Un objet **mat** a un indice de spécularité près de 0.0



Application d'un matériau

- Possible d'appliquer **un** matériau à **plusieurs objets**
- Possible d'appliquer **plusieurs** matériaux à **un objet**
- On peut avoir un matériau qui **ignore** l'effet de la lumière (*shadeless*)
- Produit un effet intéressant lorsque combiné à l'option *freestyle* de Blender: How To Create Toon Style Animation In Blender



Matériaux 3D

Il existe plusieurs autres **propriétés** des matériaux:

- **Coordonnées de texture**: transparent ou opaque
- **Transparence**: transparent ou opaque
- **Émission**: émission de lumière
- **Réfraction**: aussi appelée « densité optique »
- **Occlusion ambiante**: indique à quel point la lumière peut se rendre à un endroit
- **Cartes** (*map*): normales, relief, déplacement, spéculaire, réflectance, occlusion ambiante
- **Anisotropie**: apparence dépend de la direction
- **Rendu physique réaliste** (*physic based rendering*): basé sur les modèles physiques (surtout optique)

Nous y reviendrons **plus tard**...

Construire un modèle

Deux stratégies

À la main

- En utilisant un logiciel de **modélisation** (Blender
- Ou en apportant des **correctifs manuels** à un modèle généré/existant
- En utilisant un **moteur de jeu** (Unity, Godot)

Par programmation

- En **scriptant** à l'aide d'un logiciel
- **Exemple**: Blender
- En utilisant une **bibliothèque** graphique (par exemple, `three.js`)
- En utilisant un **moteur de jeu** (Unity, Godot)
- Directement dans **OpenGL**

Manipulation de modèles

Primitives

- **Maillage**: cube, plan, cercle, sphère, cône, cylindre, tore, singe
- **Courbe**: de Bézier, NURBS, cercle, hélice

Sélection

Un à un (*right-click*), additive (*shift*), par boîte (`B`), circulaire (`C`), lasso (*ctrl*), par connexité (`L`), ...

Modifications

- **Destructives**: transformations de base, ajout, extrusions, sous-division, coupure (*loop cut*), fusion, lier (*bridge*), suppression, ...
- **Non destructives**: transformations de base, duplication (*array*), limage (*bevel*), sous-division (*subsurface*), épaississement (*solidify*), image-miroir (*mirror*), courbature (*curve*), ...

Démonstration

Modélisation d'un objet simple en Blender

Sculpture d'un maillage

- Concevoir un **modèle** en manipulant un maillage, même s'il existe des **transformations** est souvent long et ardu
- Plusieurs logiciels (dont Blender) offrent un mode **« sculpture »** qui permet de construire des maillages de façon **plus intuitive**
- L'outil de base est une **brosse** (*brush*), qui est hautement **configurable**
- Plus adapté lorsqu'on souhaite créer des modèles **réalistes**

Sculpture: avantages/inconvénients

Avantages

- La sculpture d'un modèle offre une grande **flexibilité**
- Il s'agit d'un outil plus naturel pour les **artistes** qui ont des connaissances plus limitées en **informatique**
- Produit des résultats très **réalistes**

Inconvénient

- Les modèles produits deviennent rapidement **très volumineux** (beaucoup de sommets/arêtes/faces)
- Heureusement, il existe des techniques permettant de **réduire** le volume par la suite (appelé *retopo* en anglais).

Géométrie 3D

Points et vecteurs

- Un **point** est un triplet $P = (x, y, z) \in \mathbb{R}^3$
- Un **vecteur** est un triplet $\vec{u} = (u_1, u_2, u_3) \in \mathbb{R}^3$

Opérations sur les vecteurs

Soient $\vec{u} = (u_1, u_2, u_3)$ et $\vec{v} = (v_1, v_2, v_3)$. Soit $k \in \mathbb{R}$.

- **Norme**: $\|\vec{u}\| = \sqrt{u_1^2 + u_2^2 + u_3^2}$
- **Somme**: $\vec{u} + \vec{v} = (u_1 + v_1, u_2 + v_2, u_3 + v_3)$
- **Multiplication par un scalaire**: $k\vec{u} = (ku_1, ku_2, ku_3)$
- **Produit scalaire**: $\vec{u} \cdot \vec{v} = \langle \vec{u}, \vec{v} \rangle = u_1 v_1 + u_2 v_2 + u_3 v_3$
- **Produit vectoriel**:

$$\vec{u} \times \vec{v} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \end{vmatrix}$$

Propriétés importantes

Soient $\vec{u}$ et $\vec{v}$ deux vecteurs non nuls

- Si θ est l'**angle** entre $\vec{u}$ et $\vec{v}$, alors

$$\vec{u} \cdot \vec{v} = \|u\| \, \|v\| \cos\theta$$

- $\vec{u}$ et $\vec{v}$ sont **orthogonaux** (perpendiculaires) si et seulement si

$$\vec{u} \cdot \vec{v} = 0$$

- $\vec{u}$ et $\vec{v}$ sont **colinéaires** si et seulement si

$$\vec{u} \times \vec{v} = \vec{0}$$

- Le vecteur $\vec{u} \times \vec{v}$ et **perpendiculaire** à $\vec{u}$ et à $\vec{v}$, c'est-à-dire

$$\vec{u} \cdot \vec{w} = \vec{v} \cdot \vec{w} = 0$$

Lieux géométriques

En 3D, plusieurs **lieux géométriques** importants:

- **Linéaires**: point, droite, plan
- **Polyèdres**: cube, cône, cylindre, boule, tore
- **Courbes**: segment, courbe de Bézier, hélice, cercle
- **Surfaces**: plan, sphère, cube, cône, cylindre, tore
- **Quadriques**: paraboloïde, hyperboloïde, cône, ellipsoïde, sphère

Représentation

- En mots
- À l'aide d'**équations** ou d'**inéquation**
- À l'aide d'**ensembles**
- À l'aide d'une **fonction vectorielle**

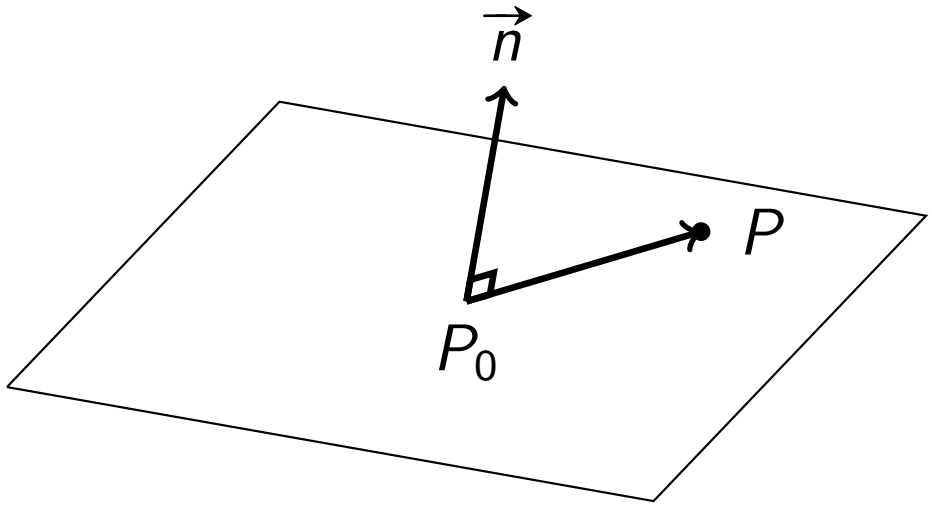
Plan

Un **plan** Π est entièrement défini par les éléments suivants:

- Un **point** P_0 et un **vecteur** non nul $\vec{n}$, appelé **vecteur normal**
- Un **point** P_0 et deux **vecteurs** non colinéaires $\vec{u}$ et $\vec{v}$

Observation

Un point P appartient à Π si et seulement si $\overrightarrow{P_0P} \cdot \vec{n} = 0$.



Équation cartésienne

Si $\vec{n} = (a, b, c)$ et $P_0 = (x_0, y_0, z_0)$, alors

$$\begin{aligned}
\Pi &= \{P \in \mathbb{R}^3 \mid \overrightarrow{P_0P} \cdot \vec{n} = 0\} \\
&= \{(x, y, z) \in \mathbb{R}^3 \mid (x - x_0, y - y_0, z - z_0) \cdot (a, b, c) = 0\} \\
&= \{(x, y, z) \in \mathbb{R}^3 \mid ax + by + cz - ax_0 - by_0 - cz_0 = 0\} \\
&= \{(x, y, z) \in \mathbb{R}^3 \mid ax + by + cz + d = 0\},
\end{aligned}$$

où $d = \vec{n} \cdot \overrightarrow{P_0O} = -ax_0 - by_0 - cz_0$.

Ainsi, l'**équation cartésienne** d'un plan est de la forme

$$ax + by + cz + d = 0$$

où (a, b, c) est normal au plan.

Questions sur les plans

- Soient Π_1 et Π_2 deux **plans**
- Définis respectivement par les **vecteurs normaux** $\vec{n_1}$ et $\vec{n_2}$
- Et passant respectivement par les **points** P_1 et P_2

1. Comment peut-on décider si Π_1 et Π_2 sont **confondus**?

Questions sur les plans

- Soient Π_1 et Π_2 deux **plans**
- Définis respectivement par les **vecteurs normaux** $\vec{n_1}$ et $\vec{n_2}$
- Et passant respectivement par les **points** P_1 et P_2

1. Comment peut-on décider si Π_1 et Π_2 sont **confondus**?

$\rightarrow$ On vérifie si $\vec{n_1}$ et $\vec{n_2}$ sont **colinéaires**

$\rightarrow$ Puis si P_1 **appartient** à Π_2

2. S'ils sont **parallèles**?

Questions sur les plans

- Soient Π_1 et Π_2 deux **plans**
- Définis respectivement par les **vecteurs normaux** $\vec{n_1}$ et $\vec{n_2}$
- Et passant respectivement par les **points** P_1 et P_2

1. Comment peut-on décider si Π_1 et Π_2 sont **confondus**?

$\rightarrow$ On vérifie si $\vec{n_1}$ et $\vec{n_2}$ sont **colinéaires**

$\rightarrow$ Puis si P_1 **appartient** à Π_2

2. S'ils sont **parallèles**?

$\rightarrow$ On vérifie si $\vec{n_1}$ et $\vec{n_2}$ sont **colinéaires**

$\rightarrow$ Puis si P_1 **n'appartient** à Π_2

3. S'ils sont **sécants**?

Questions sur les plans

- Soient Π_1 et Π_2 deux **plans**
- Définis respectivement par les **vecteurs normaux** $\vec{n_1}$ et $\vec{n_2}$
- Et passant respectivement par les **points** P_1 et P_2

1. Comment peut-on décider si Π_1 et Π_2 sont **confondus**?

$\rightarrow$ On vérifie si $\vec{n_1}$ et $\vec{n_2}$ sont **colinéaires**

$\rightarrow$ Puis si P_1 **appartient** à Π_2

2. S'ils sont **parallèles**?

$\rightarrow$ On vérifie si $\vec{n_1}$ et $\vec{n_2}$ sont **colinéaires**

$\rightarrow$ Puis si P_1 **n'appartient** à Π_2

3. S'ils sont **sécants**?

$\rightarrow$ On vérifie si $\vec{n_1}$ et $\vec{n_2}$ ne sont pas **colinéaires**

4. S'ils sont sécants, quelle est leur **intersection**?

Questions sur les plans

- Soient Π_1 et Π_2 deux **plans**
- Définis respectivement par les **vecteurs normaux** $\vec{n_1}$ et $\vec{n_2}$
- Et passant respectivement par les **points** P_1 et P_2

1. Comment peut-on décider si Π_1 et Π_2 sont **confondus**?

$\rightarrow$ On vérifie si $\vec{n_1}$ et $\vec{n_2}$ sont **colinéaires**

$\rightarrow$ Puis si P_1 **appartient** à Π_2

2. S'ils sont **parallèles**?

$\rightarrow$ On vérifie si $\vec{n_1}$ et $\vec{n_2}$ sont **colinéaires**

$\rightarrow$ Puis si P_1 **n'appartient** à Π_2

3. S'ils sont **sécants**?

$\rightarrow$ On vérifie si $\vec{n_1}$ et $\vec{n_2}$ ne sont pas **colinéaires**

4. S'ils sont sécants, quelle est leur **intersection**?

$\rightarrow$ C'est une **droite** de **vecteur directeur** $\vec{n_1} \times \vec{n_2}$

$\rightarrow$ Pour le **point**, il faut résoudre un système d'équations

Droite et segment

Une **droite** Δ est entièrement définie par les éléments suivants:

- Un **point** P_0 et un **vecteur** non nul $\vec{u}$
- Deux **points** distincts P_0 et P_1
- Par l'**intersection** de deux plan **sécants**

Par exemple, c'est un ensemble de la forme

$$\Delta = \{P \in \mathbb{R}^3 \mid P = P_0 + k\vec{u}, k \in \mathbb{R}\}$$

Le **segment** S qui relie les points P_0 et P_1 est donné par

$$S = \{P \in \mathbb{R}^3 \mid P = P_0 + k(P_1 - P_0), \quad 0 \leq k \leq 1\}$$

Une **paramétrisation** de S est donc

$$\begin{array}{rcl} \vec{r}(t) : [0,1] & \rightarrow & \mathbb{R}^3 \\ t & \mapsto & P_0 + t(P_1 - P_0) \end{array}$$

Questions sur les droites

Soient Δ_1 et Δ_2 deux droites de vecteurs directeurs $\vec{u_1}$ et $\vec{u_2}$ passant par les points P_1 et P_2 respectivement

- Quelles sont les **relations** possibles entre Δ_1 et Δ_2?

Questions sur les droites

Soient Δ_1 et Δ_2 deux droites de vecteurs directeurs $\vec{u_1}$ et $\vec{u_2}$ passant par les points P_1 et P_2 respectivement

- Quelles sont les **relations** possibles entre Δ_1 et Δ_2?

$\rightarrow$ *confondues*: elles se croisent en **tout point**

$\rightarrow$ *parallèles*: elles ont la même **direction**

$\rightarrow$ *sécantes*: elles se croisent en **un point**

$\rightarrow$ *gauches*: elles ne se **croisent pas** et sont **non-parallèles**

- Comment peut-on **déduire** la relation?

Questions sur les droites

Soient Δ_1 et Δ_2 deux droites de vecteurs directeurs $\vec{u_1}$ et $\vec{u_2}$ passant par les points P_1 et P_2 respectivement

- Quelles sont les **relations** possibles entre Δ_1 et Δ_2?

→ *confondues*: elles se croisent en **tout point**

→ *parallèles*: elles ont la même **direction**

→ *sécantes*: elles se croisent en **un point**

→ *gauches*: elles ne se **croisent pas** et sont **non-parallèles**

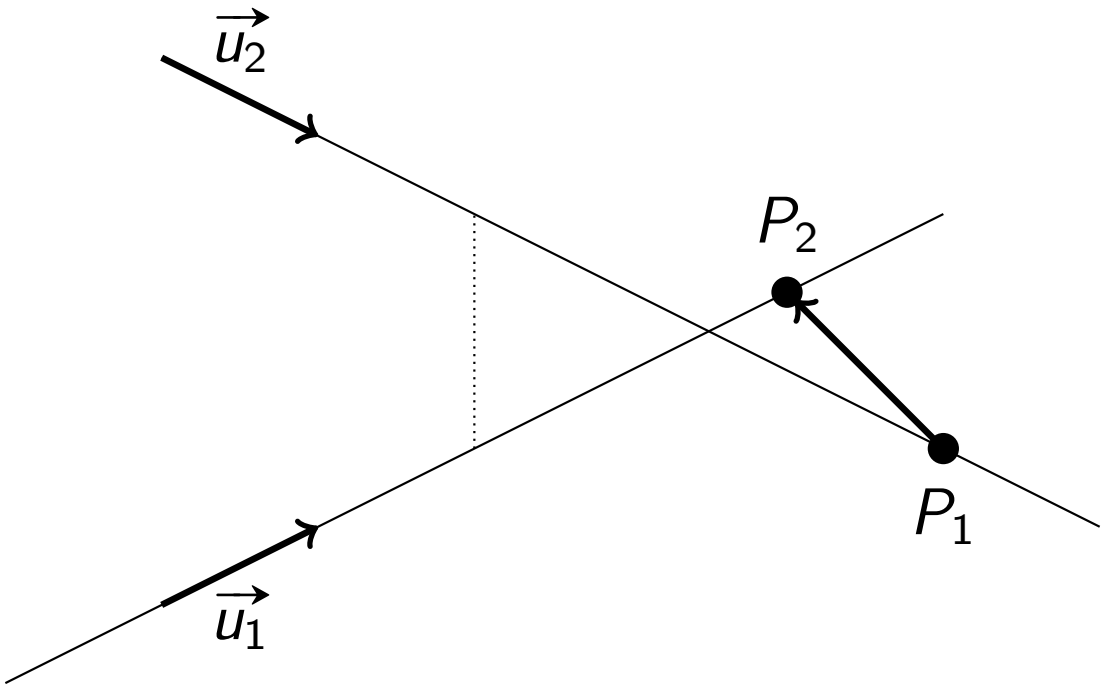
- Comment peut-on **déduire** la relation?

→ Est-ce que les vecteurs sont **colinéaires**?

→ **Si oui**, alors on vérifie si elles partagent un point (confondues) ou non (parallèles)

→ **Sinon**, on vérifie si les droites sont dans un même plan (sécantes) ou non (gauches)

Intersection de deux droites (1/3)



Intersection de deux droites (2/3)

- On vérifie si P_1 se trouve sur la droite Δ_2 et si P_2 se trouve sur la droite Δ_1
- Si oui, alors les droites sont **confondues** ou alors on a **un point d'intersection**
- Sinon, on considère les **produits vectoriels** suivants

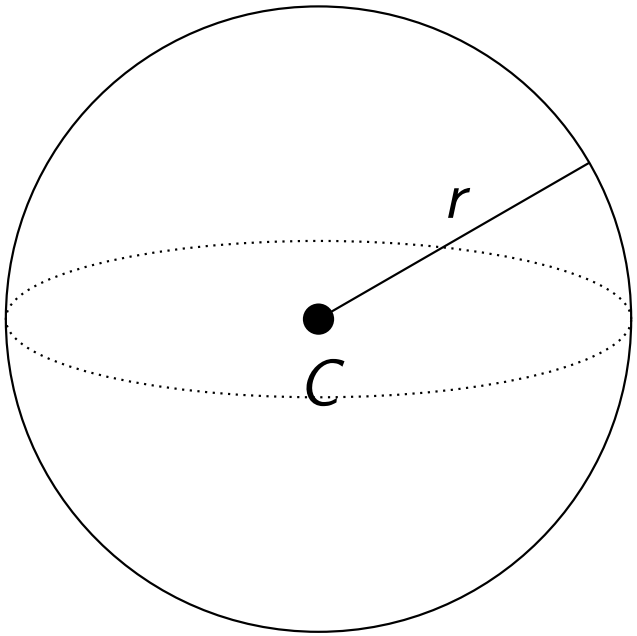
$$\vec{u_1} \times \vec{u_2} \quad \text{et} \quad \vec{u_1} \times \overrightarrow{P_1 P_2}$$

- Si les deux vecteurs ne sont pas dans la même **direction**, alors les droites sont **gauches**
- Sinon, c'est que les droites se trouvent dans un **même plan**, donc elles s'**intersectent**!

Intersection de deux droites (3/3)

1: **fonction** INTERSECTION(Δ_1, Δ_2 : droites)
2: **si** Δ_1.point $\in \Delta_2$ **et** Δ_2.point $\in \Delta_1$ **alors**
3: **retourner** Δ_1 ▷ Droites confondues
4: **sinon**
5: $\vec{u} \leftarrow \Delta_2$.point $- \Delta_1$.point
6: $\vec{v} \leftarrow \Delta_1$.dir $\times \vec{u}$
7: $\vec{w} \leftarrow \Delta_2$.dir $\times \Delta_1$.dir
8: **si** $\vec{v}$ et $\vec{w}$ ne sont pas dans la même direction **alors**
9: **retourner** *rien* ▷ Pas d'intersection
10: **fin si**
11: $s \leftarrow 1$ si $\vec{v}$ et $\vec{w}$ ont la même orientation, -1 sinon
12: **retourner** Δ_1.point $+ \frac{s\|\vec{v}\|}{\|\vec{w}\|} \Delta_1$.dir ▷ Unique point
13: **fin si**
14: **fin fonction**

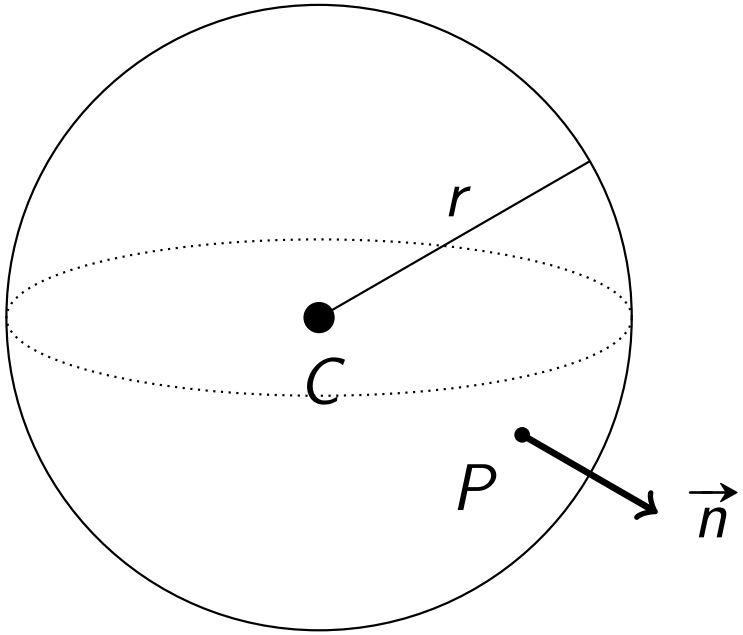
Sphère



- Soit S une **sphère** de centre $C = (h, k, \ell)$ et de rayon r
- Un point $P = (x, y, z)$ se trouve **sur** S si et seulement si $\left\|\overrightarrow{CP}\right\|^2 = r^2$, c'est-à-dire si

$$(x-h)^2 + (y-k)^2 + (z-\ell)^2 = r^2$$

Vecteur normal d'une sphère

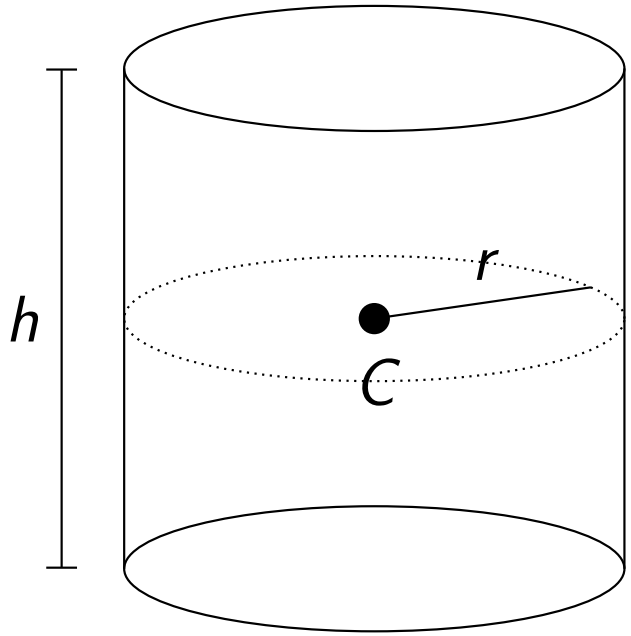


Si P est un point de S, alors le vecteur

$$\vec{n} = \frac{\overrightarrow{CP}}{\left\| \overrightarrow{CP} \right\|}$$

est **normal** à S au point P

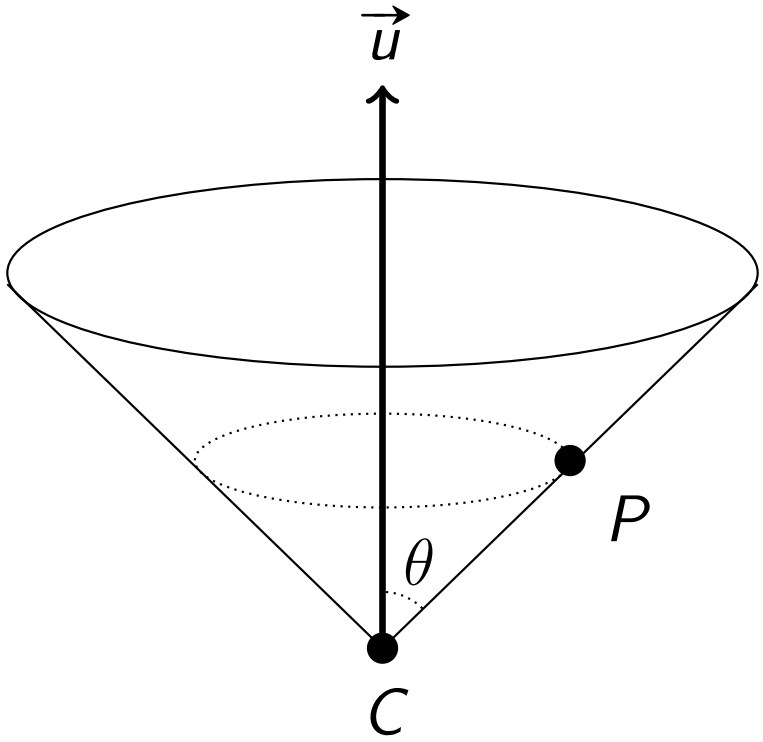
Cylindre



Un point $P = (x, y, z)$ se trouve **sur**

- le **contour** du cylindre si et seulement si $x^2 + y^2 = r^2$ et $-h/2 \leq z \leq h/2$
- une des **bases** si et seulement si $x^2 + y^2 \leq r^2$ et $z \in \{-h/2, h/2\}$

Cône



- Soit $P = (x, y, z)$ un point **sur** le cône
- Soit θ le **demi-angle** d'ouverture du cône
- Alors on doit avoir

$$\vec{u} \cdot \overrightarrow{CP} = \|\vec{u}\| \left\|\overrightarrow{CP}\right\| \cos\theta$$

Surfaces paramétrées

Surfaces paramétrées

Définition

- Soit S une surface
- On dit que la fonction vectorielle

$$\vec{s}(u, v) = (x(u, v), y(u, v), z(u, v)), \quad (u, v) \in D$$

est une **paramétrisation** de la surface S si

$$\text{Image}(\vec{s}) = S$$

- L'ensemble D est appelé **domaine** des paramètres u et v;

Remarque

Trouver une **paramétrisation** d'une surface revient à **inverser** un **développement** de cette surface.

Changements de coordonnées

- Un autre type de fonctions très importantes en infographie: les **changements de coordonnées**
- L'idée est qu'il est plus facile de décrire **certaines surfaces** en utilisant un **repère** différent

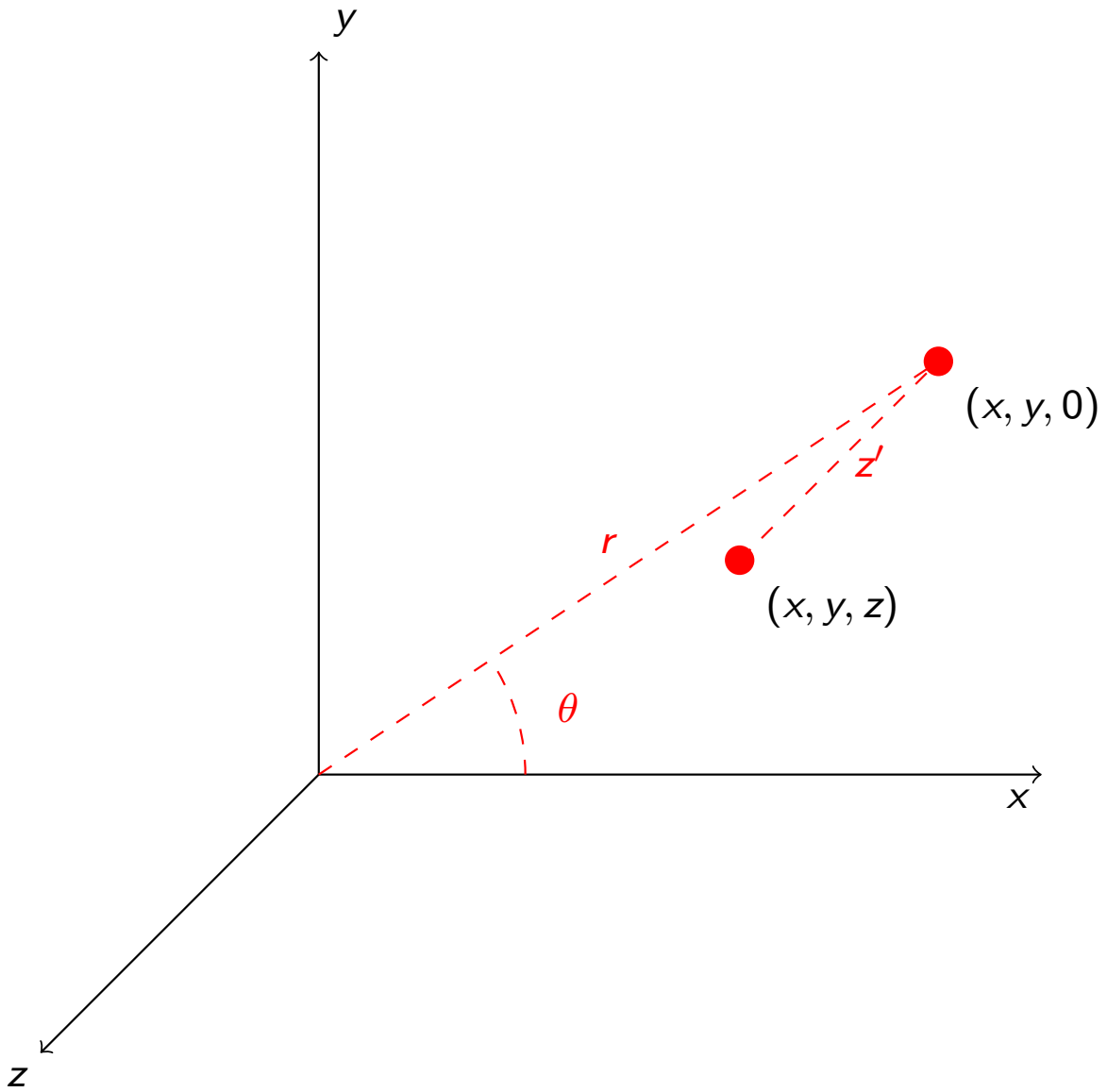
Systèmes les plus courants

- Les coordonnées **cartésiennes**
- Les coordonnées **polaires**
- Les coordonnées **cylindriques**
- Les coordonnées **sphériques**

Remarque

On peut imaginer une **infinité** d'autres changements.

Les coordonnées cylindriques (1/2)



Les coordonnées cylindriques (2/2)

- C'est une généralisation des coordonnées polaires en **3D**
- **Cartésiennes**: (x, y, z)
- **Cylindriques**: (r, θ, z')

$\rightarrow$ r : distance par rapport à l'axe des z

$\rightarrow$ θ : angle entre les vecteurs $(1, 0, 0)$ et $(x, y, 0)$

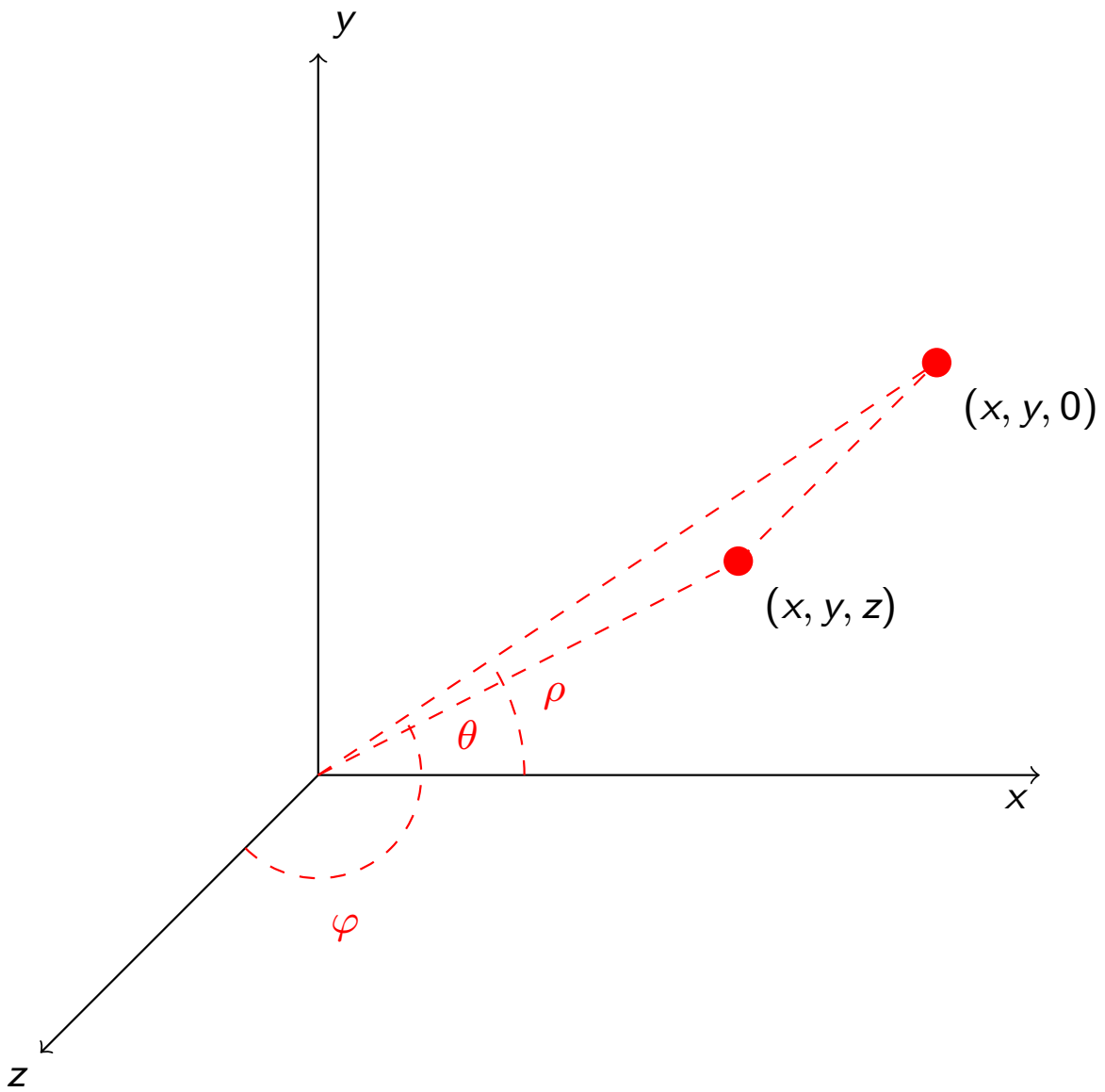
$\rightarrow$ z' : distance (signée) par rapport au plan xy

- Équations:

$$x = r\cos\theta, \quad y = r\sin\theta, \quad z = z'.$$

- Peut être construit à partir des **autres axes** aussi

Les coordonnées sphériques (1/2)



Les coordonnées sphériques (2/2)

- Une autre généralisation des coordonnées polaires en **3D**
- **Cartésiennes**: (x, y, z)
- **Sphériques**: (ρ, θ, φ)

$\rightarrow$ ρ : distance par rapport à l'origine

$\rightarrow$ θ : angle entre les vecteurs $(1, 0, 0)$ et $(x, y, 0)$

$\rightarrow$ φ : angle entre les vecteurs $(0, 0, 1)$ et (x, y, z)

- Équations:

$$x = \rho \sin\theta \cos\varphi, \quad y = \rho \sin\theta \sin\varphi, \quad z = \rho \cos\theta.$$

Question sur les changements de coordonnées

- Trouver une paramétrisation du **plan**

$$2x - y + 3z - 5 = 0.$$

- Trouver une paramétrisation du **cylindre**

$$y^2 + z^2 = 25.$$

- Trouver une paramétrisation de la **sphère**

$$x^2 + y^2 + z^2 = r^2.$$

Représenter un modèle

Formats de fichier

Il existe plusieurs **formats** de fichier pour représenter des **maillages**:

- **OBJ**: développé initialement par Wavefront Technologies, format textuel simple, relativement bien supporté, relativement limité
- **DAE** = *Digital Asset Exchange*, aussi appelé COLLADA, format XML, maintenu par le groupe Khronos, à privilégier pour le transfert de Blender vers Godot
- **FBX** = *FilmBoX*, initié par Kaydara (maintenant dans Autodesk), format propriétaire, à privilégier pour le transfert de Blender vers Unity
- **3DS** (Autodesk, 3DS Max), etc.

Les trois derniers format possèdent **plus d'options**: squelette, animations, physique, etc.

Le format OBJ

- Format **textuel** de base, lisible par l'humain
- Développé par **Wavefront Technologies** (racheté par **Alias** et ensuite **Autodesk**)
- Format **ouvert** et assez **universellement** reconnu
- Spécification: ici

Syntaxe

- **Commentaire**: ligne commençant par `#`
- **Sommet**: ligne commençant par `v`
- **Coordonnée de texture**: ligne commençant par `vt`
- **Vecteur normal**: ligne commençant par `vn`
- **Face**: ligne commençant par `f`
- **Objet**: ligne commençant par `o`
- **Lissage**: lignes commençant par `s`, etc.

Le format MTL

- Format **compagnon** du format OBJ
- Permet de préciser les **propriétés** du matériau du modèle
- Spécification: ici

Syntaxe

- Couleur **ambiante**: `Ka <r> <g> <b>`
- Couleur **diffuse**: `Kd <r> <g> <b>`
- Couleur **spéculaire**: `Ks <r> <g> <b>`
- **Exposant** de spécularité: `Ns <valeur>`
- Couleur d'**émission**: `Ke <r> <g> <b>`
- Indice de **réfraction**: `Ni <valeur>`
- **Opacité**: `d <valeur>`
- **Illumination**: `illum <code>`, etc.

Exemple: cube par défaut de Blender

Fichier `cube.obj`

```
# Blender v2.79 (sub 0) OBJ File: 'cube.blend'
# www.blender.org
mtllib cube.mtl
o Cube
v 1.000000 -1.000000 -1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 -1.000000 1.000000
v -1.000000 -1.000000 -1.000000
v 1.000000 1.000000 -0.999999
v 0.999999 1.000000 1.000001
v -1.000000 1.000000 1.000000
v -1.000000 1.000000 -1.000000
vn 0.0000 -1.0000 0.0000
vn 0.0000 1.0000 0.0000
vn 1.0000 0.0000 0.0000
vn -0.0000 -0.0000 1.0000
vn -1.0000 -0.0000 -0.0000
vn 0.0000 0.0000 -1.0000
usemtl Material
s off
f 1//1 2//1 3//1 4//1
f 5//2 8//2 7//2 6//2
f 1//3 5//3 6//3 2//3
f 2//4 6//4 7//4 3//4
f 3//5 7//5 8//5 4//5
f 5//6 1//6 4//6 8//6
```

Fichier `cube.mtl`

```
# Blender MTL File: 'cube.blend'
# Material Count: 1

newmtl Material
Ns 96.078431
Ka 1.000000 1.000000 1.000000
Kd 0.640000 0.640000 0.640000
Ks 0.500000 0.500000 0.500000
Ke 0.000000 0.000000 0.000000
Ni 1.000000
d 1.000000
illum 2
```

Le format DAE

- Aussi appelé **COLLADA**
- Maintenu par le groupe Khronos (qui supporte entre autres OpenGL)
- Format textuel basé sur **XML**
- Format beaucoup plus **riche**
- Permet d'inclure des informations sur les **matériaux**, la **physique**, les **animations**, etc.
- Exemple: `cube.dae`
- Se manipule rarement **directement**: mieux vaut utiliser une bibliothèque (par exemple Pycollada)
- **Remarque**: si vous utilisez Godot, assurez-vous d'utiliser leur **script d'exportation** plutôt que celui disponible dans Blender par défaut

Génération procédurale

Génération procédurale

Plusieurs types

- Génération de **modèles**
- Génération de **textures**
- Génération de **matériaux** procéduraux
- Génération de **scènes** ou de **cartes**
- Système de **particules**

Scripts

- Blender supporte l'utilisation de **scripts**
- Le langage utilisé est **Python**
- On peut appeler un script **depuis l'interface**
- On peut aussi l'appeler en **ligne de commande**

Génération d'un tétraèdre (1/2)

```
 1  import bpy
 2  from math import pi, cos, sin
 3
 4  def cylindrical_to_cartesian(r, theta, z):
 5      return (r * cos(theta), r * sin(theta), z)
 6
 7  # Vertices and faces
 8  vertices = [cylindrical_to_cartesian(1.0, 0.0 * pi / 3.0, 0.0),
 9              cylindrical_to_cartesian(1.0, 2.0 * pi / 3.0, 0.0),
10              cylindrical_to_cartesian(1.0, 4.0 * pi / 3.0, 0.0),
11              (0.0, 0.0, 1.0)]
12  faces = [(0, 2, 1),
13           (0, 1, 3),
14           (1, 2, 3),
15           (2, 0, 3)]
16
17  # Model
18  mesh_data = bpy.data.meshes.new('Tetrahedron')
19  mesh_data.from_pydata(verts, [], faces)
20  mesh_data.update()
21  obj = bpy.data.objects.new('Tetrahedron', mesh_data)
22
23  # Scene
24  scene = bpy.context.scene
25  scene.objects.link(obj)
26  obj.select = True
27  bpy.ops.wm.save_as_mainfile(filepath='tetrahedron.blend')
```

Génération d'un tétraèdre (2/2)

Commande

```
$ blender --background --python tetrahedron.py
```

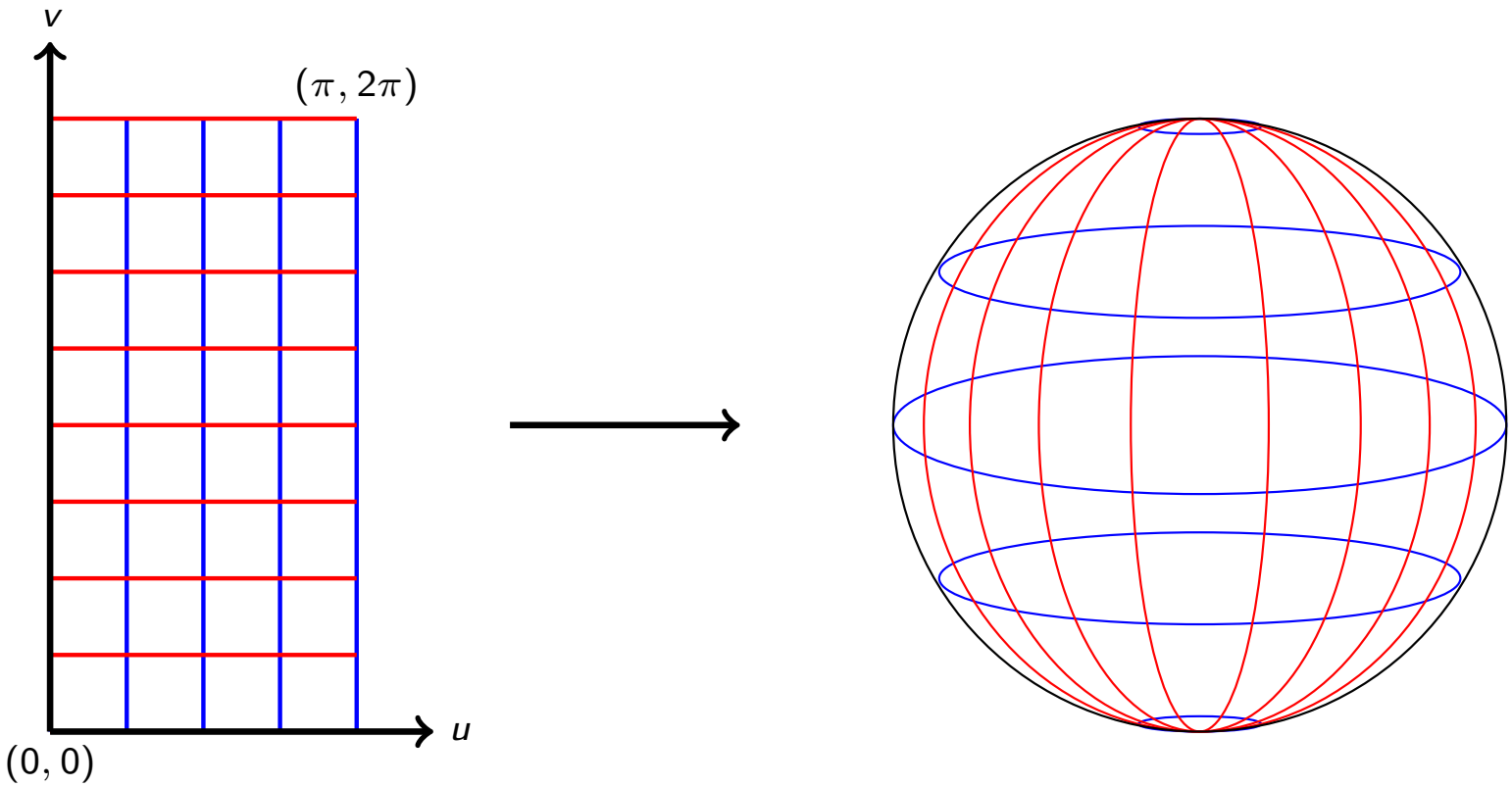
Options

- `--background` empêche l'ouverture de l'**interface graphique**
- `--python scene.py` passe le **script** `scene.py` qui sera exécuté

Aide

```
$ blender --help
```

Génération d'une sphère (1/3)



$$\vec{s}(u, v) = (\sin u \cos v, \sin u \sin v, \cos u)$$

Génération d'une sphère (2/3)

```
 1  import bpy
 2  from math import pi, cos, sin
 3
 4  def spherical_to_cartesian(rho, theta, phi):
 5      r"""
 6      Returns the cartesian coordinates from the spherical ones.
 7      """
 8      return (rho * sin(theta) * cos(phi),\
 9              rho * sin(theta) * sin(phi),\
10              rho * cos(theta))
11
12  def vertex_index(i, j):
13      r"""
14      Returns the vertex index from `(i,j)`
15      """
16      return (i % m) + (j % (p + 1)) * m
17
18  # Vertices and faces
19  m = 32
20  p = 16
21  ustep = pi / p
22  vstep = 2.0 * pi / m
23  verts = [spherical_to_cartesian(1.0, i * ustep, j * vstep)\
24      for i in range(p + 1) for j in range(m)]
25  faces = [(vertex_index(i, j),          vertex_index(i + 1, j),
26            vertex_index(i + 1, j + 1), vertex_index(i, j + 1))\
27      for i in range(m + 1) for j in range(p + 1)]
```

Génération d'une sphère (3/3)

```
28
29  # Model
30  mesh_data = bpy.data.meshes.new('Sphere')
31  mesh_data.from_pydata(verts, [], faces)
32  mesh_data.update()
33  obj = bpy.data.objects.new('Sphere', mesh_data)
34
35  # Scene
36  scene = bpy.context.scene
37  scene.objects.link(obj)
38  obj.select = True
39  bpy.ops.wm.save_as_mainfile(filepath='sphere.blend')
```

```
$ blender --background --python sphere.py
```

Génération d'une scène (1/2)

```
1  import bpy
2  import random
3
4  # Cleaning default scene
5  bpy.ops.object.select_all(action='DESELECT')
6  bpy.data.objects['Camera'].select = True
7  bpy.data.objects['Cube'].select = True
8  bpy.data.objects['Lamp'].select = True
9  bpy.ops.object.delete()
10
11 # Materials
12 ground_material = bpy.data.materials.new('GroundMaterial')
13 ground_material.diffuse_color = (0, 0.1, 0)
14 trunk_material = bpy.data.materials.new('TrunkMaterial')
15 trunk_material.diffuse_color = (0.3, 0.05, 0)
16 leaves_material = bpy.data.materials.new('LeavesMaterial')
17 leaves_material.diffuse_color = (0, 0.5, 0)
18
19 # Creating ground
20 bpy.ops.mesh.primitive_cube_add(location=(0, 0, 0))
21 bpy.context.active_object.name = 'Ground'
22 bpy.context.scene.objects.active.scale = (8, 8, 0.1)
23 bpy.context.active_object.data.materials.append(ground_material)
```

Génération d'une scène (2/2)

```
25  # Creating trees
26  locations = [tuple(random.randint(-7, 7) for _ in range(2))\
27                                           for _ in range(10)]
28  for (i,(x,y)) in enumerate(locations):
29      bpy.ops.mesh.primitive_uv_sphere_add(size=0.8, location=(x,y,2))
30      bpy.context.active_object.name = 'Leaves.%s' % i
31      bpy.context.active_object.data.materials.append(leaves_material)
32      bpy.ops.mesh.primitive_cylinder_add(radius=0.1, location=(x,y,1))
33      bpy.context.active_object.name = 'Tree.%s' % i
34      bpy.context.active_object.data.materials.append(trunk_material)
35      bpy.data.objects['Leaves.%s' % i].select = True
36      bpy.ops.object.join()
37
38  # Saving the scene
39  bpy.ops.wm.save_as_mainfile(filepath='scene.blend')
```

```
$ blender --background --python scene.py
```

Systèmes de particules

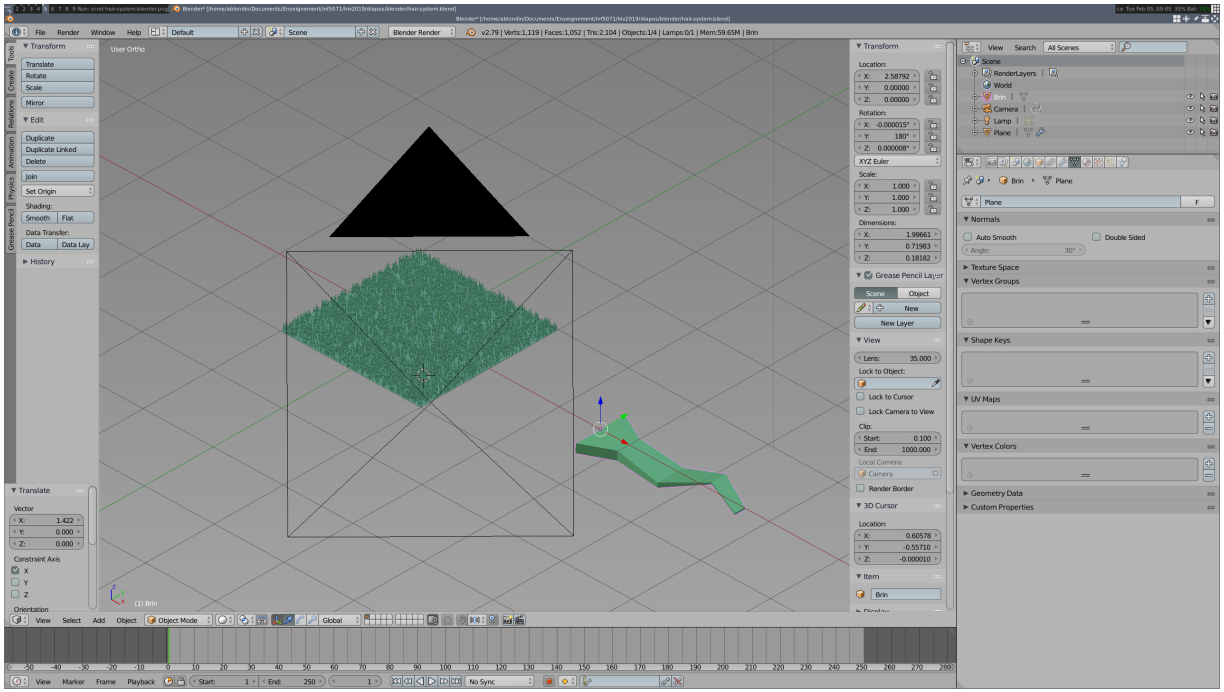
Qu'est-ce que c'est?

- **Particule** = un modèle (généralement simple)
- **Objectif**: produire plusieurs **copies** de la particule
- Possibilité de faire **varier** le modèle
- Possibilité de contrôler ses **paramètres**
- Utile pour des modeles **statiques** (*hair system*)
- Mais éventuellement aussi pour des **animations**: fumée, feu, explosions, etc.

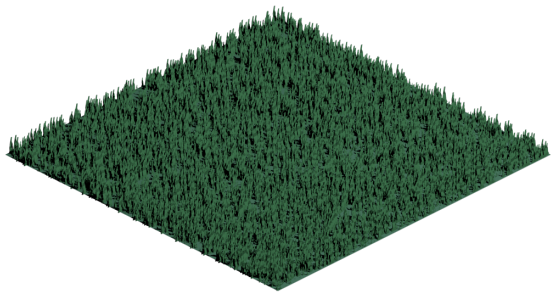
Où les trouve-t-on?

- Dans la plupart des logiciels de **modélisation**
- Et dans les **moteurs** de jeux

Démonstration



Brin et système de particules



Rendu