Introduction à Unity
Troisième partie
Version 4.6

Guillaume Levieux

Conservatoire National des Arts et Métiers

28 janvier 2015

Déroulé

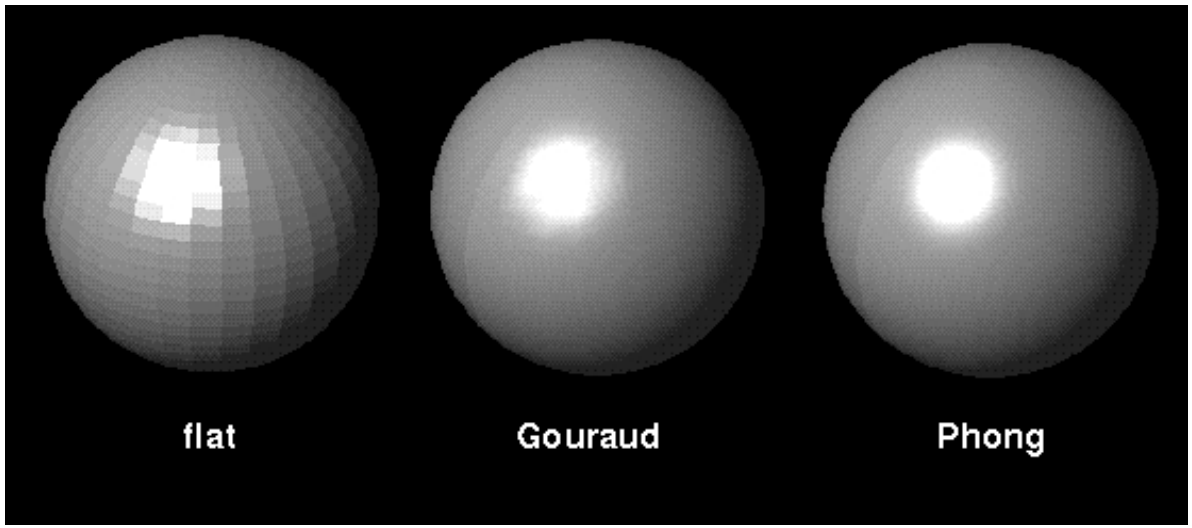
Les materials

Les materials

Principe du material

Un material est un aggregation de propriétés de rendu d'une surface.
Définit avec les données et le calcul du rendu du material.

- Un material peut utiliser des textures
- Il définit l'interaction avec la lumière (diffuse, speculaire)
- Il est lié à un programme shader (qui définit le calcul)

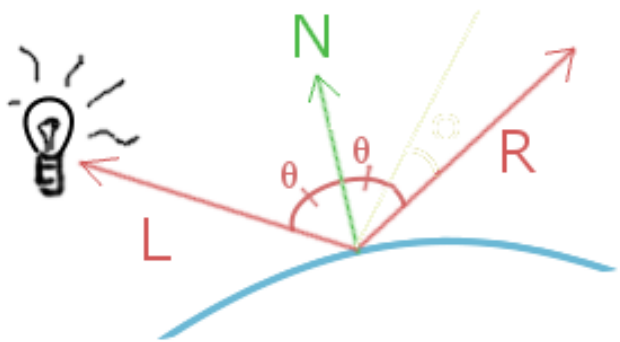


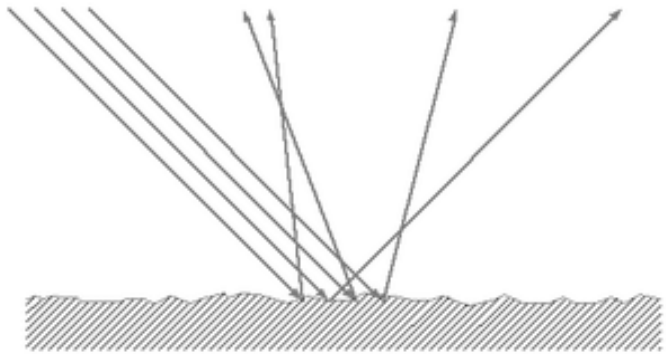
Lumière diffuse

Utilise le modèle lambertien

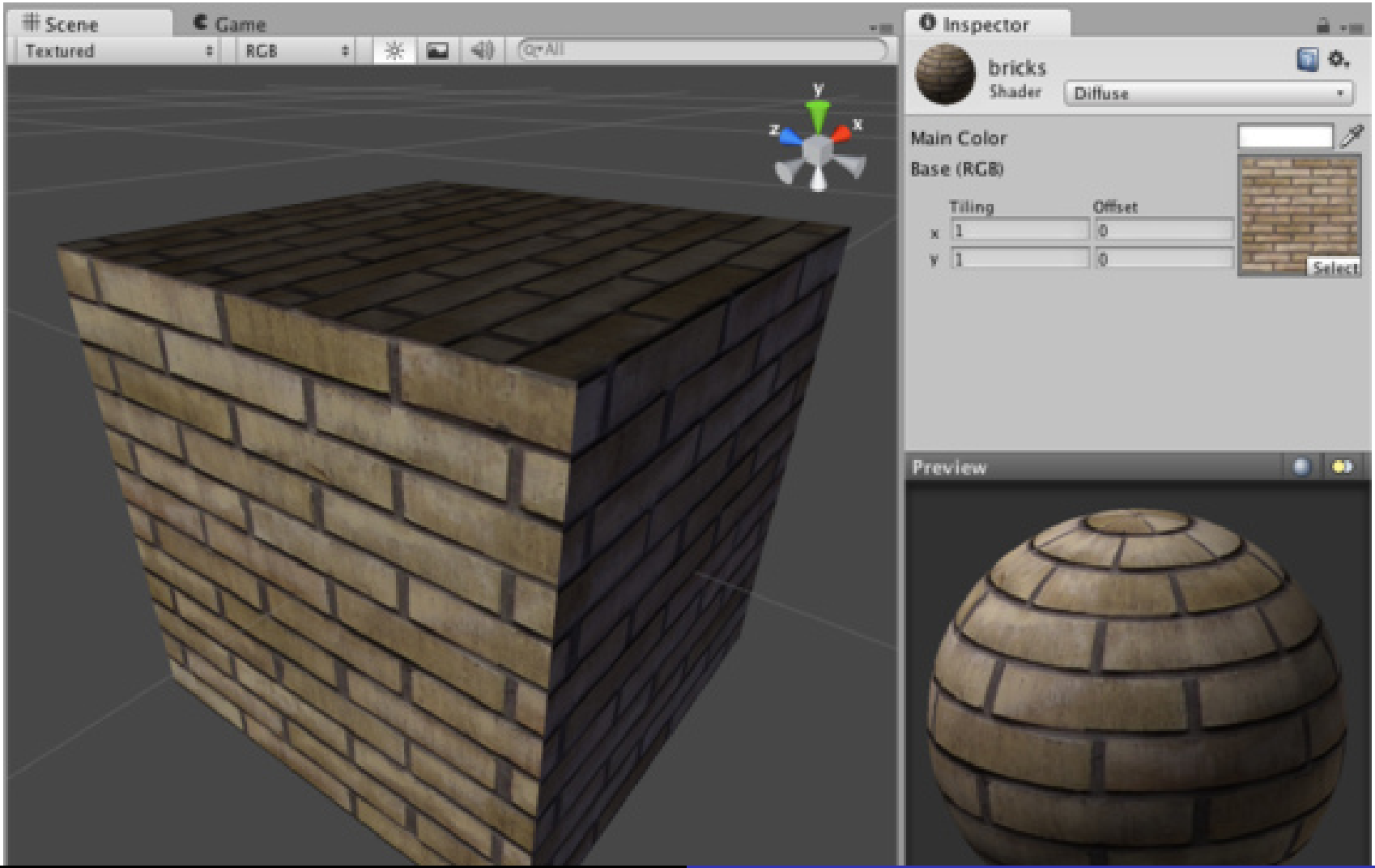
L'éclairage d'un point est uniquement dépendant de :

- la position de la lumière
- la valeur de la normale en ce point
- la couleur de la lumière
- la couleur du materiau (absorption d'une partie du spectre)





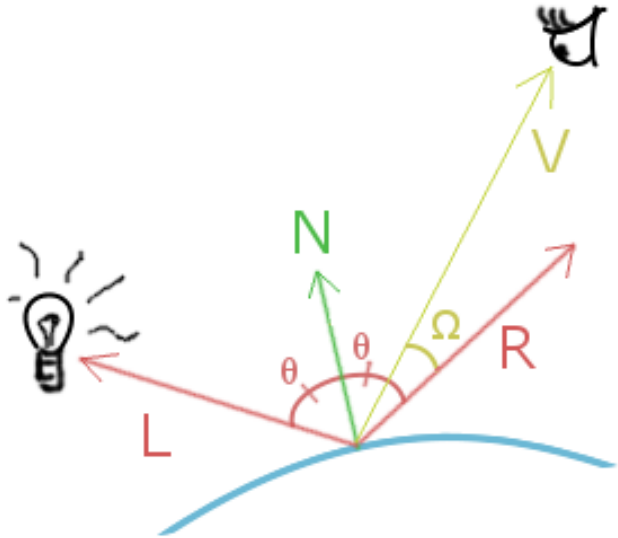
Lumière diffuse

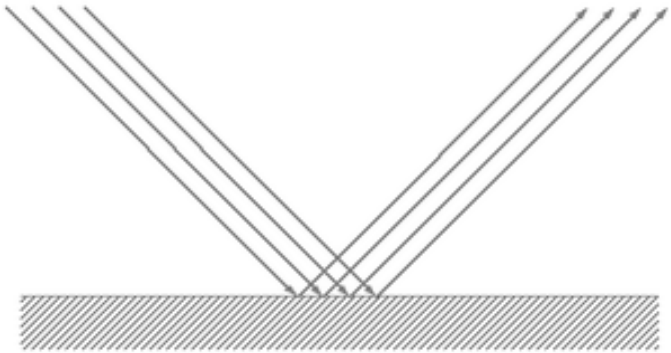


Lumière spéculaire

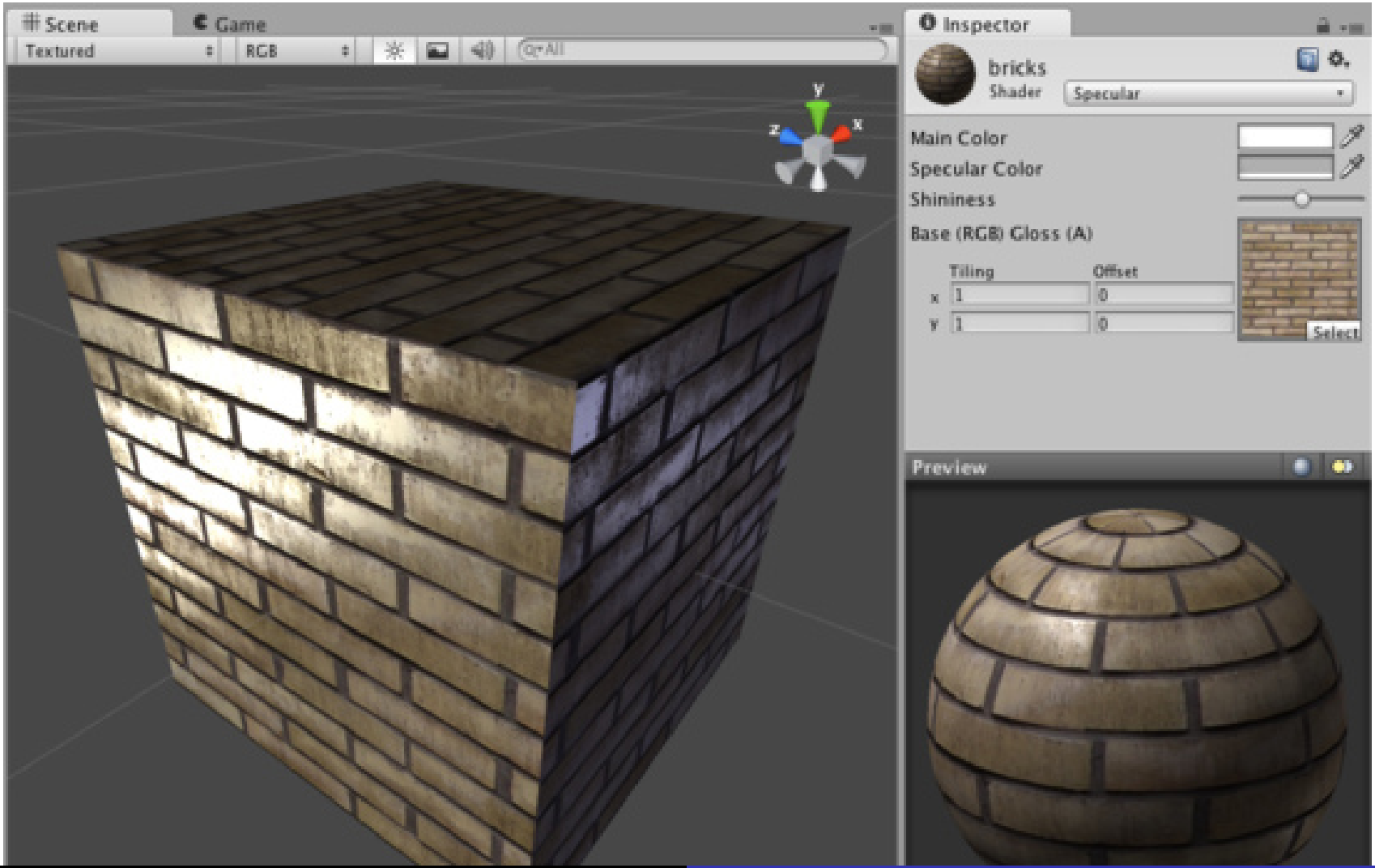
Utilise le modèle lambertien + reflet spéculaire = blinn-phong
L'éclairage d'un point dépend :

- des données du modèle lambertien (diffus)
- de la position de la camera : cone de réflexion de la source.





Lumière spéculaire

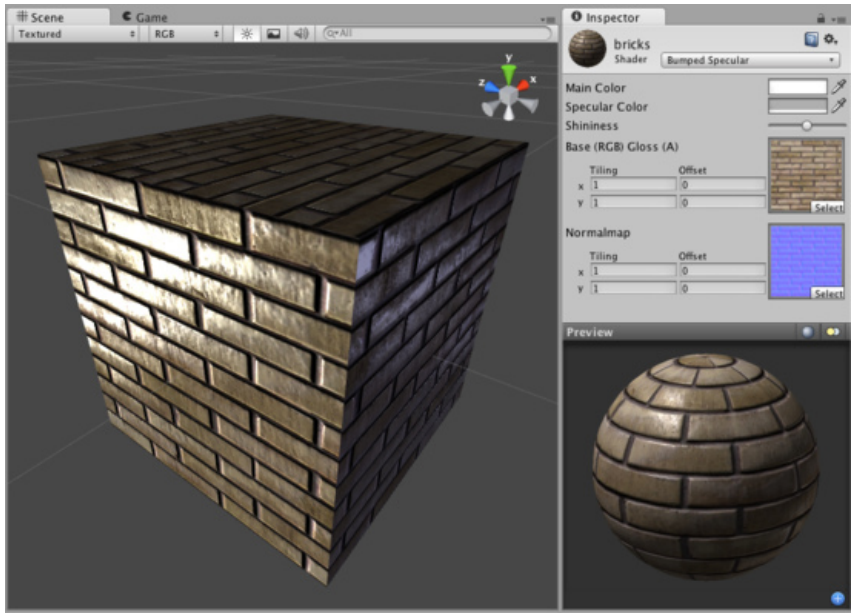


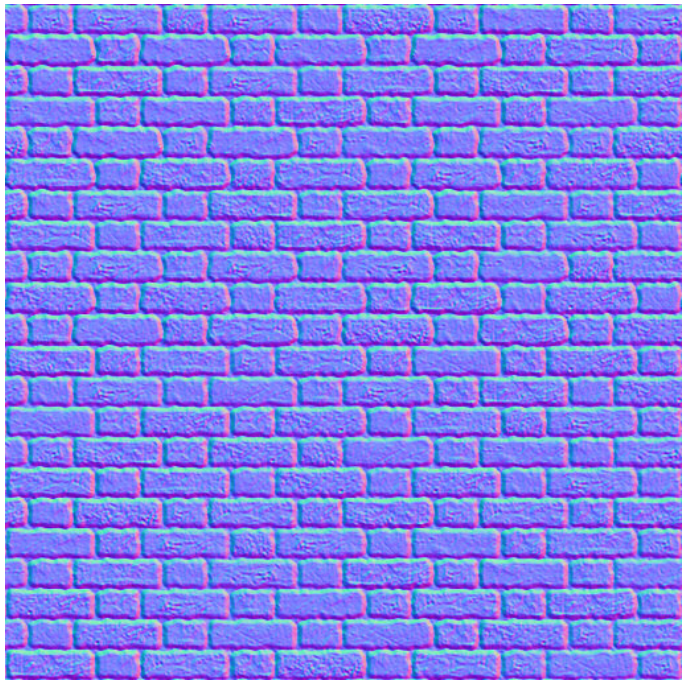
Bump mapping

Perturbation de la normale grâce à une texture

Simule un relief qui n'existe pas :

- Donne un impression de relief
- Demande un calcul de lumière de type phong (par pixel)



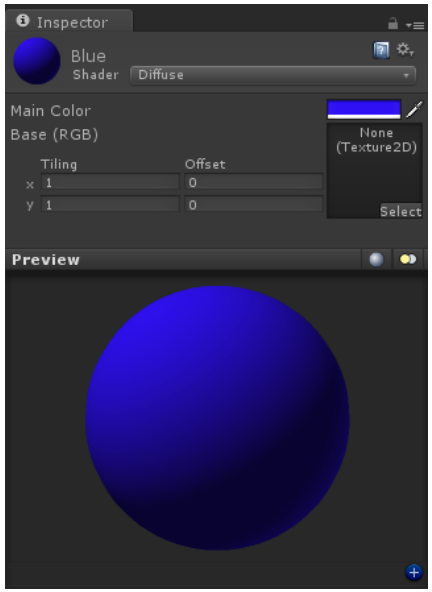


Creation d'un material

Dans la project window : Create -> Material

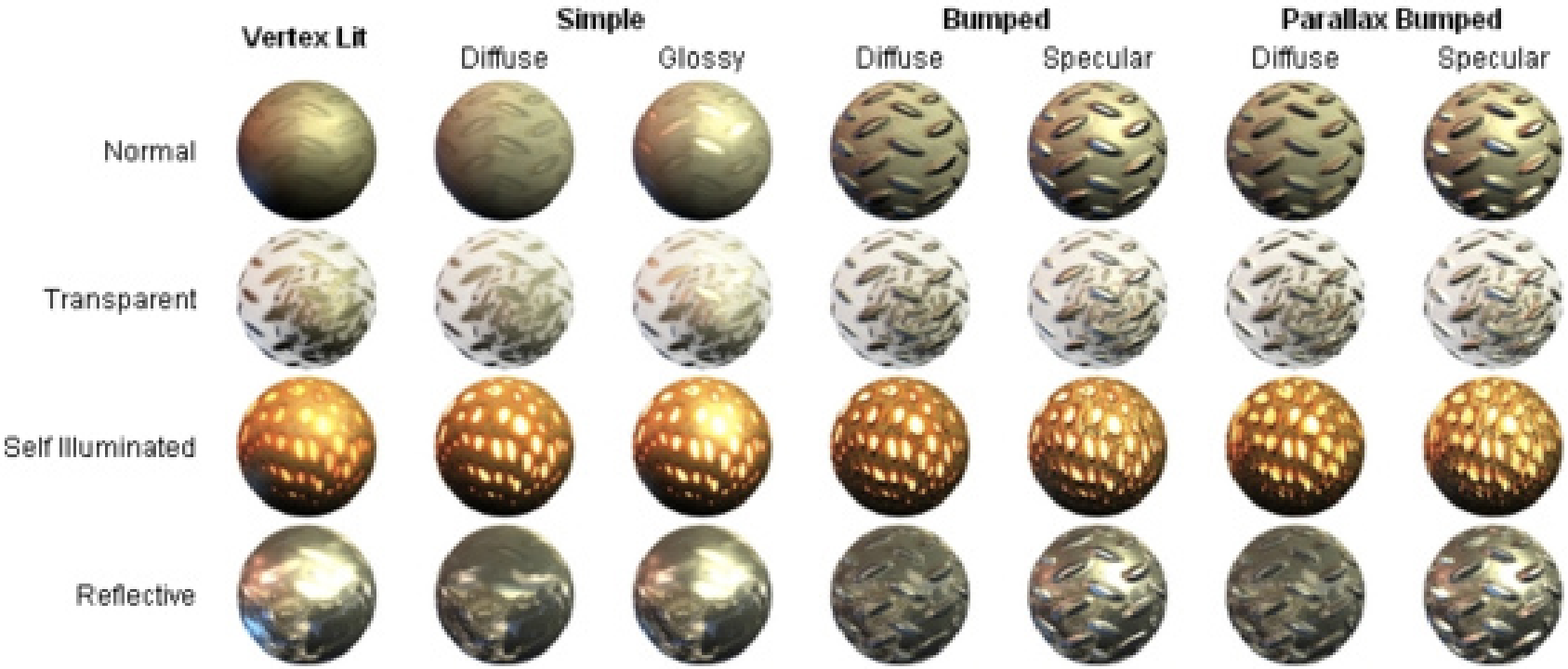
L'inspector permet de :

- Choisir le shader
- Choisir les couleurs (dont composante alpha)
- Choisir les textures + (tiling et offset)
- Voir une preview (volume de rendu et éclairage modifiables)



Plusieurs types de shaders

On peut sélectionner plusieurs shaders de base :

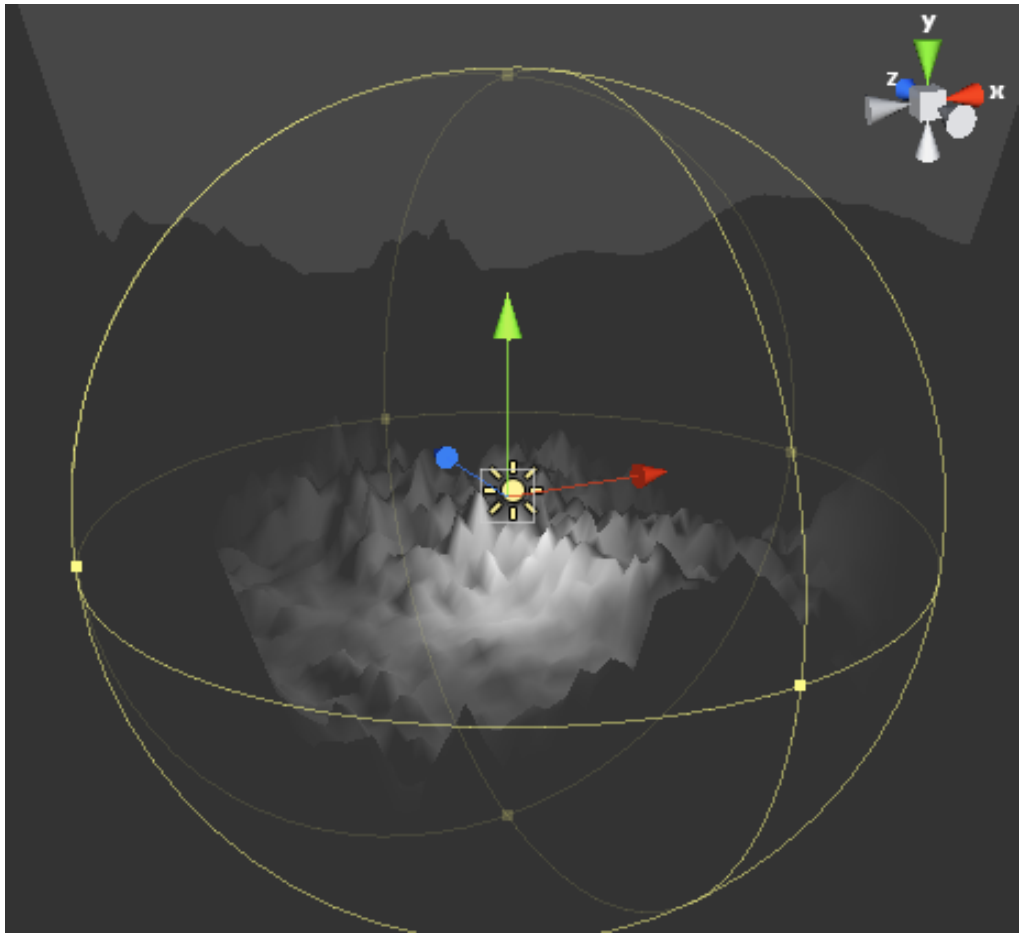


Les lumières

Les lumières

Les point lights

Lumière omni-directionnelle placée en un point



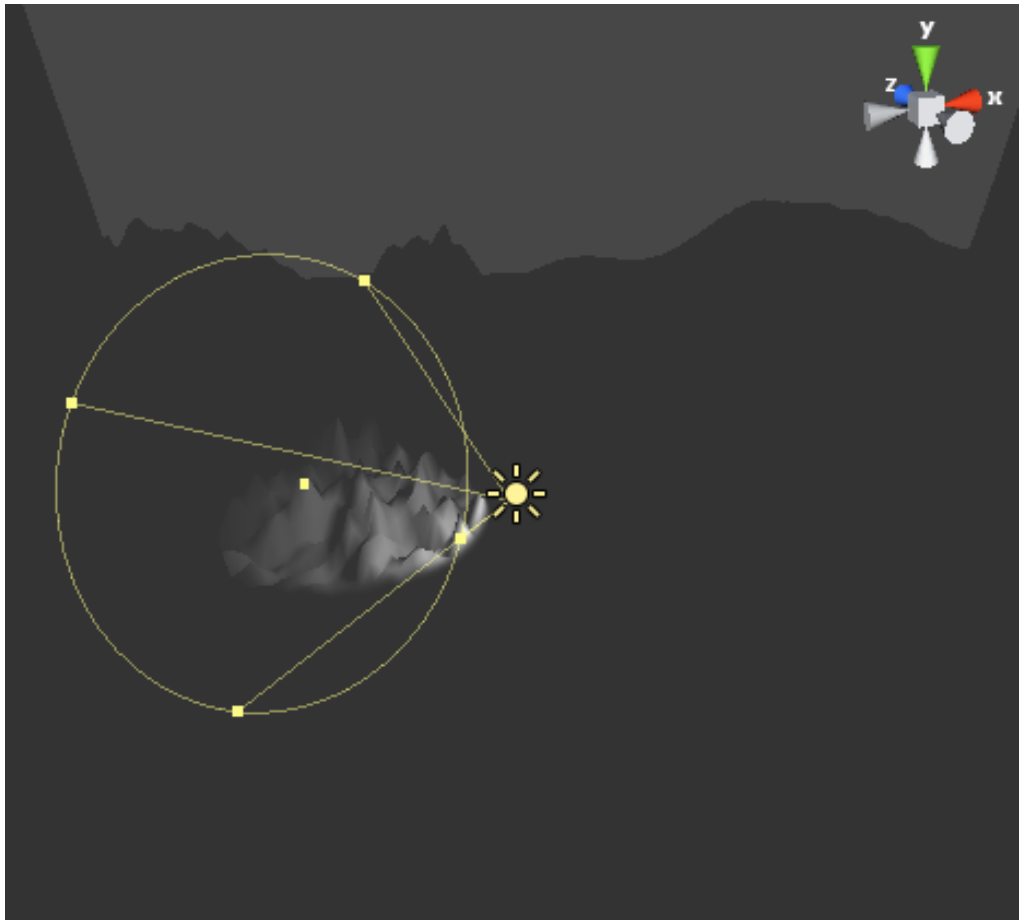
Les point lights

Une point light a :

- Un rayon d'action (range)
- Une couleur
- Une intensité
- Par defaut : pas d'ombres (mais possible par shader)
- Rappel : pas de lumière blanche
- Flare et cookie

Les spot lights

Comme une point light mais contrainte à un cone de lumière



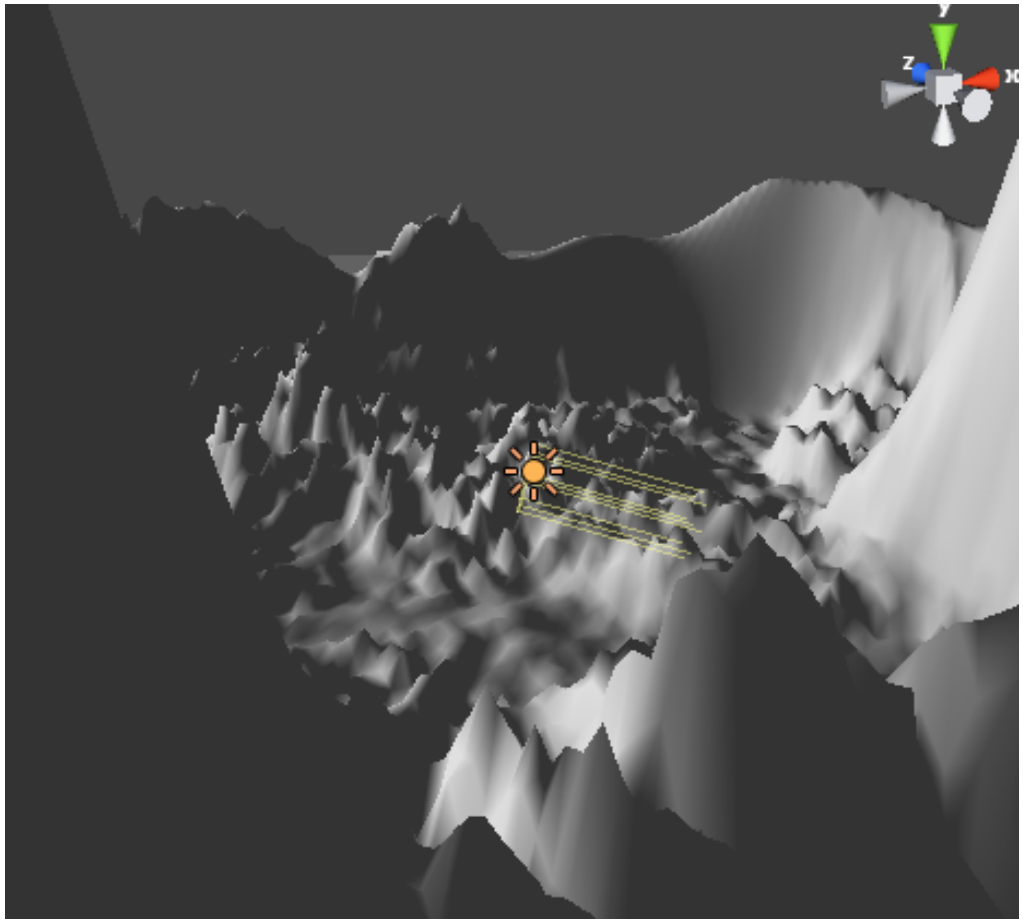
Les spot lights

Une spot light a :

- Un rayon d'action (range)
- Un largeur de spot
- Une couleur
- Une intensité
- Par defaut : pas d'ombres (mais possible par shader)
- Flare et cookie

Les directional lights

Comme le soleil : source infinie et rayons paralelles : juste une direction



Les spot lights

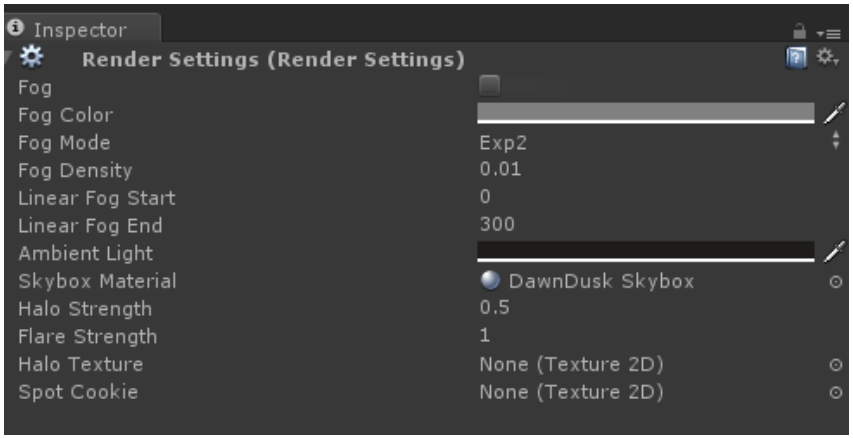
Une spot light a :

- Une couleur
- Une intensité
- Projette hard et soft shadows
- Flare et cookie

La lumière ambiante

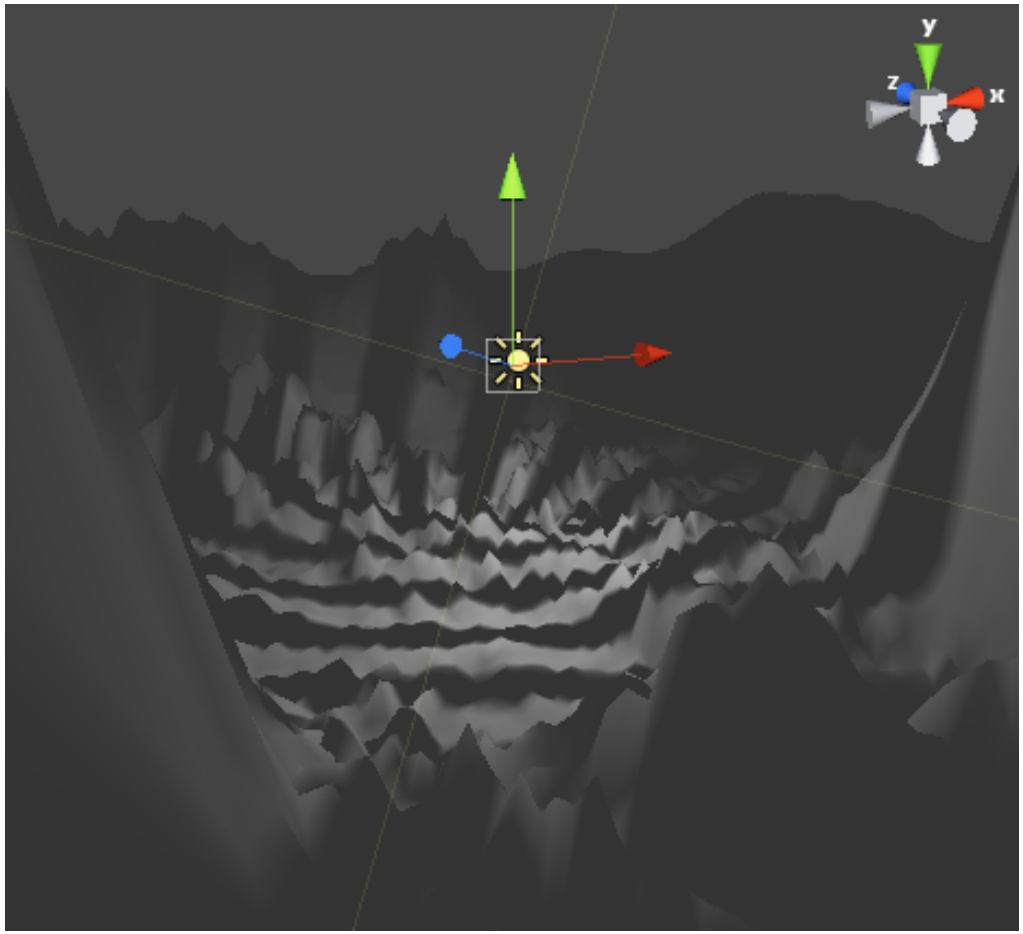
Moteur de rendu limité :

- Ne sait pas calculer les reflexions
- Certains objets sont completements noirs (irréaliste)
- On utilise une valeur de lumière ambiante
- Ajoutée à tout volume de la scène
- Se modifie dans les render settings (Edit)



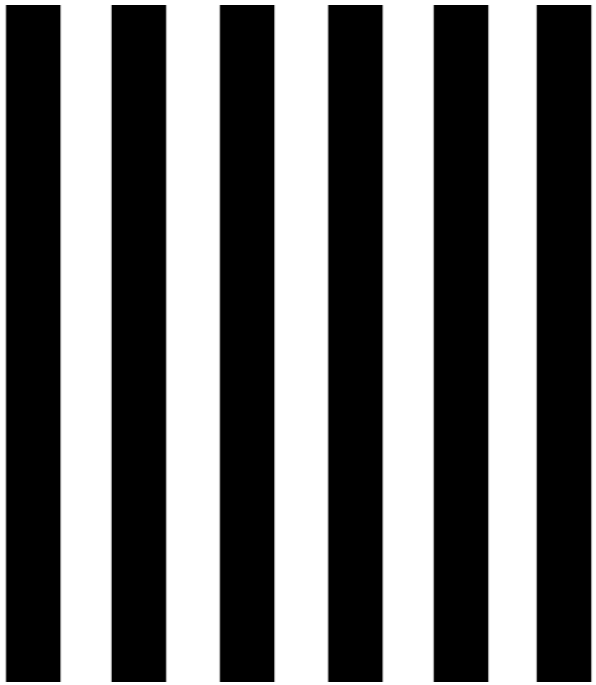
Les cookies

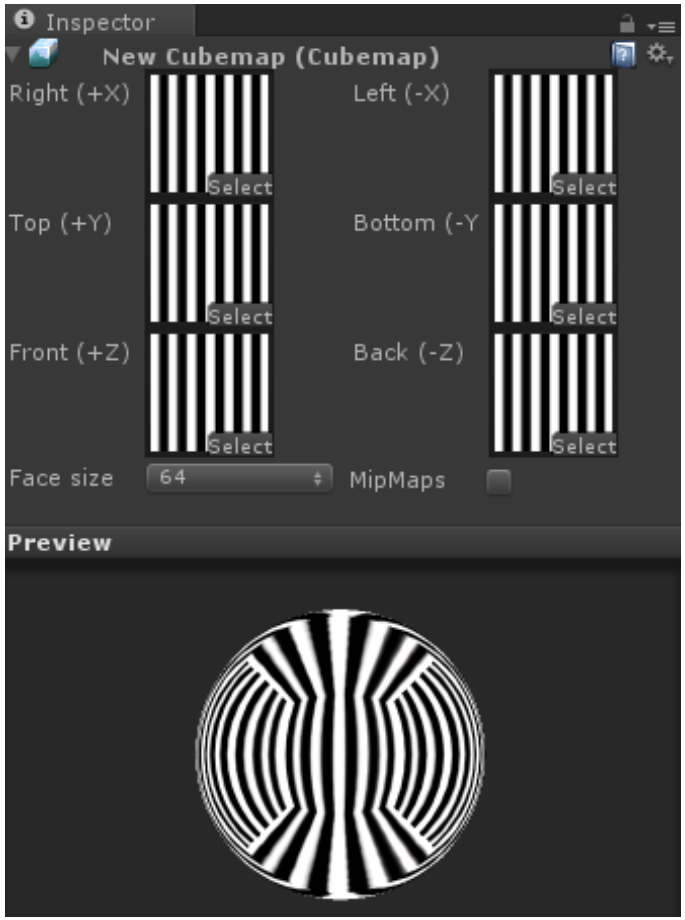
Applique une texture à la lumière



Sur une spot light

Utilise un cube map

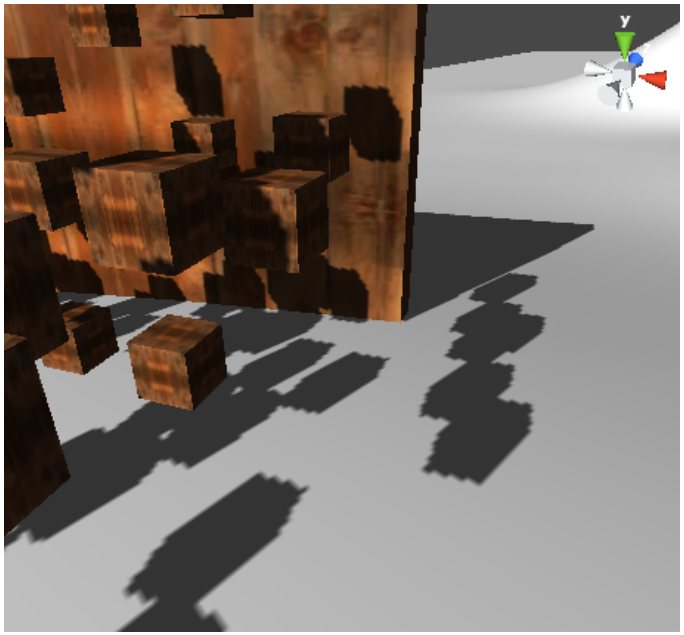




Hard et Soft shadows

Ombre plus ou moins nette

Soft demande plus de temps de calcul



Les skybox

Les skybox

La skybox

Permet d'avoir un ciel à l'infini :

- Une box ou sont plaquées des 6 textures de ciel
- Se déplace avec la caméra : impression d'infini
- 9 skybox de base livrées (material)
- s'applique dans les render settings (Edit)

Création d'un terrain

Création d'un terrain

Les terrains

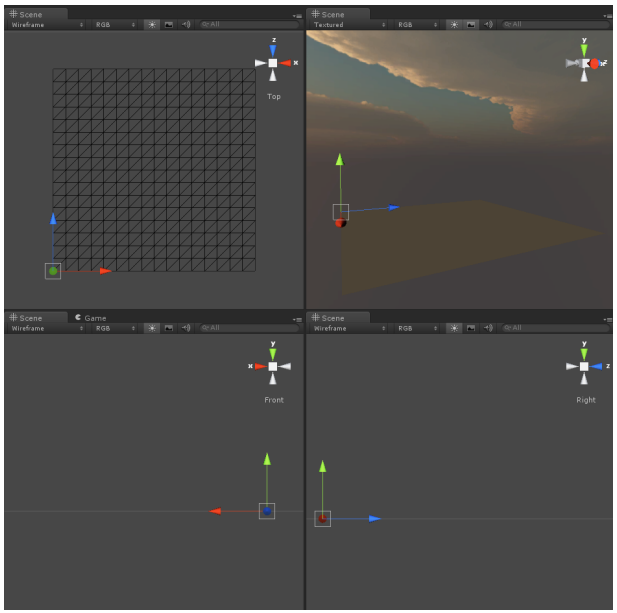
Unity possède son propre système de gestion de terrains

- Surface d'élévation
- Niveau de détail automatique
- Outils pour gérer l'élévation
- Outils pour gérer les textures
- Outil de gestion de la végétation

Les terrains

Pour créer un terrain

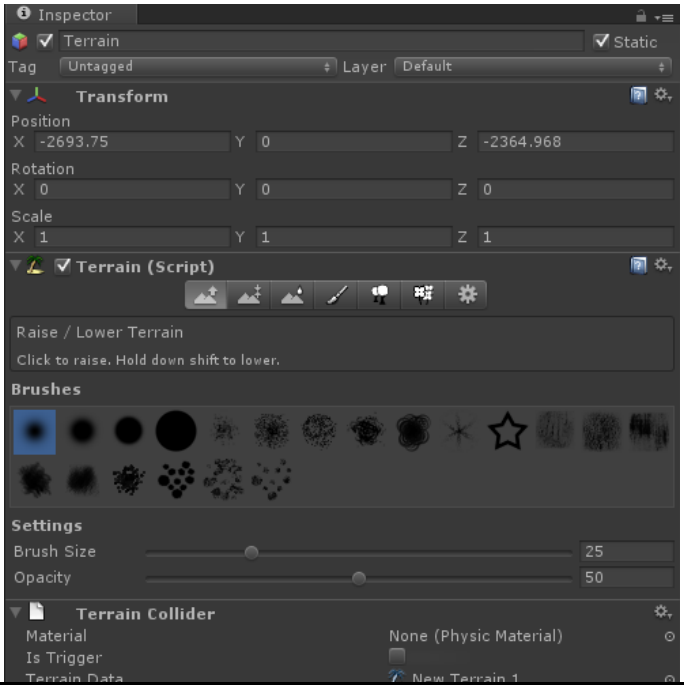
- Menu terrain
- Create terrain
- Par defaut : 2000m / 2000m
- Modifiable : Terrain -> Set Resolution
- Le translater ou on le souhaite

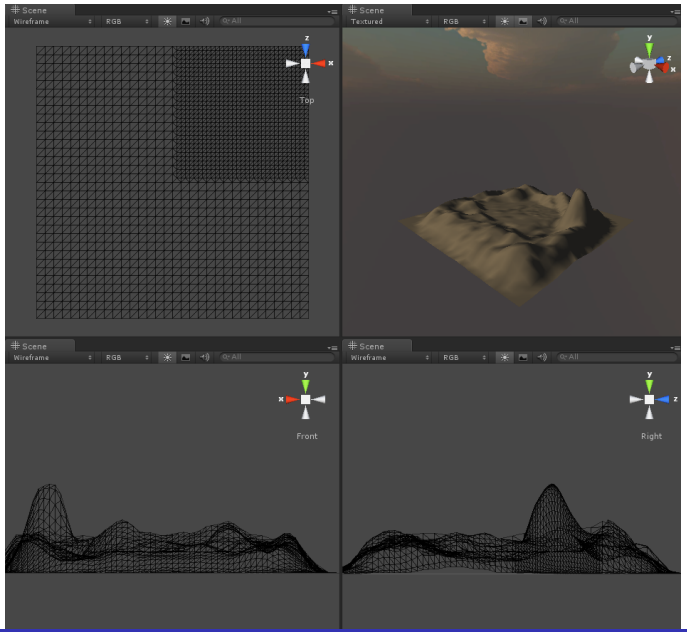


Les terrains

Plusieurs outils de modelage du terrain

- On utilise un pinceau
- On peut élever / abaisser le terrain
- On peut égaliser le terrain
- On peut niveler le terrain

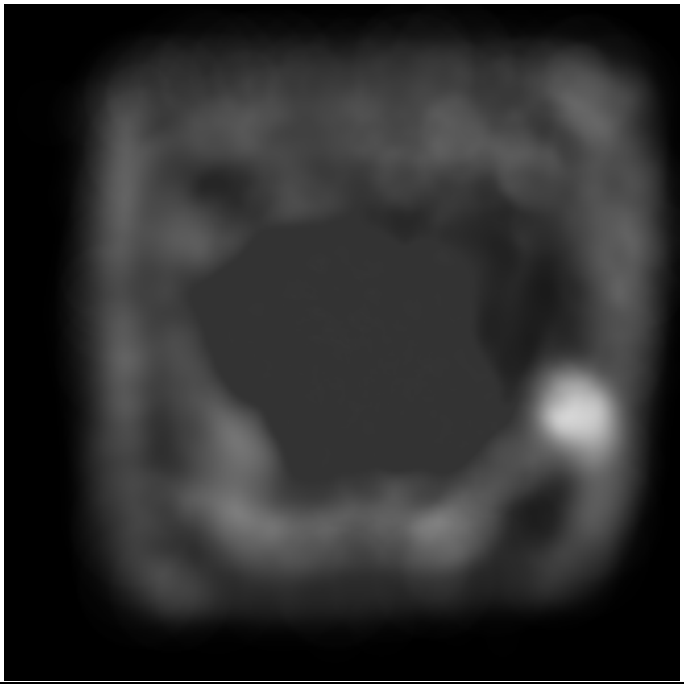


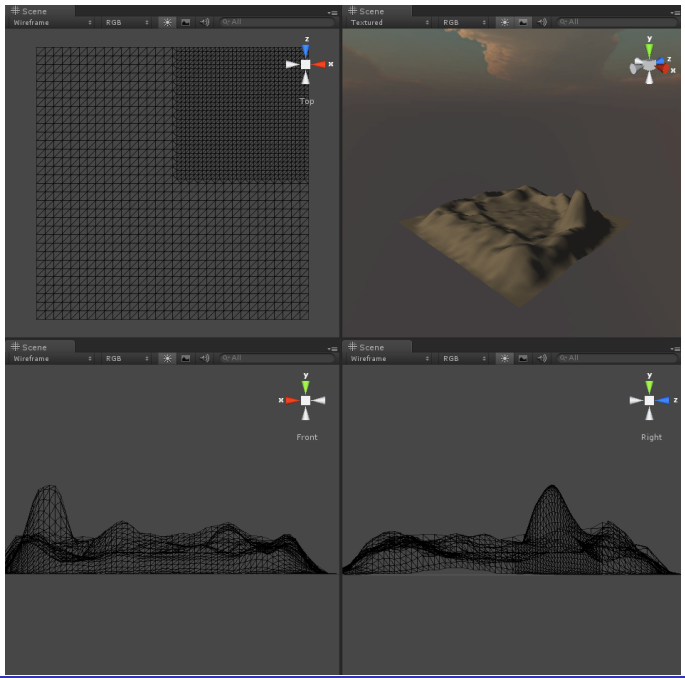


Les terrains

Possibilité d'importer / exporter la heightmap :

- Sous forme de texture
- Fichier .raw
- Editable sous photoshop
- Et réimportable

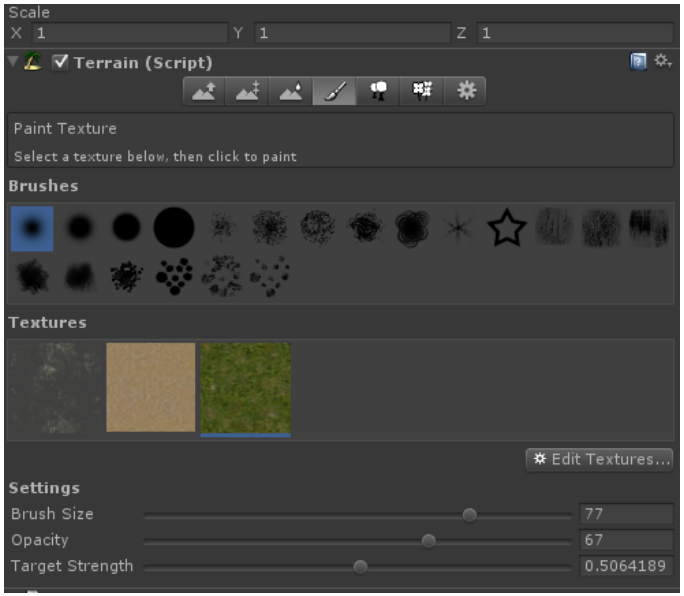


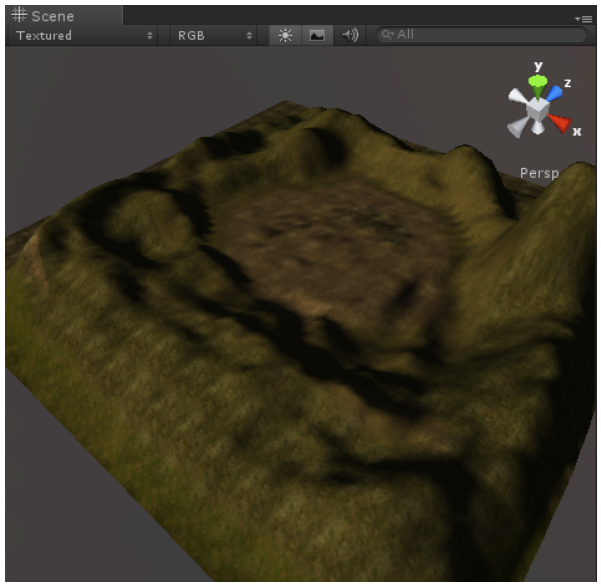


Les terrains

Le terrain peut être ensuite multi-texturé

- Au pinceau
- Charger des couches de textures (Add texture)
- Choisir la texture à appliquer
- Peindre pour ajouter la texture
- Blend avec le "Target Strength"

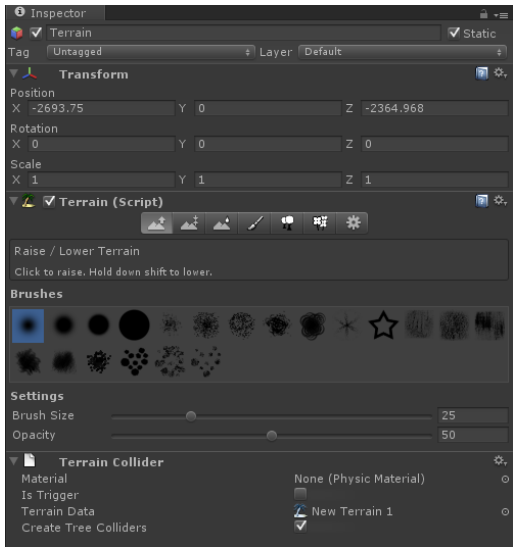


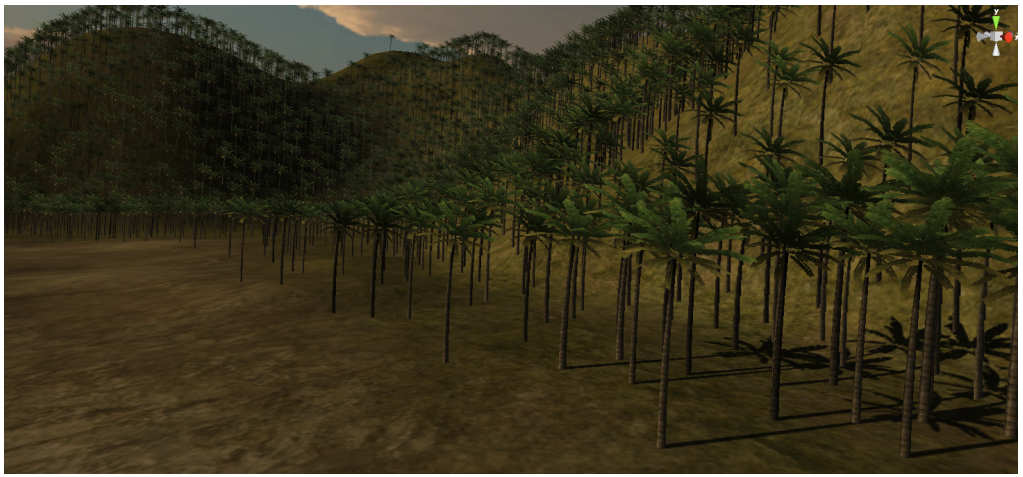


Les terrains

Unity propose un outil d'ajout de vegetation

- Au pinceau
- Un outil pour les arbres
- Un pour l'herbe et les détails
- Comme pour les textures : add et paint
- On peut ajouter les arbres en mass Terrain->Mass Place Trees

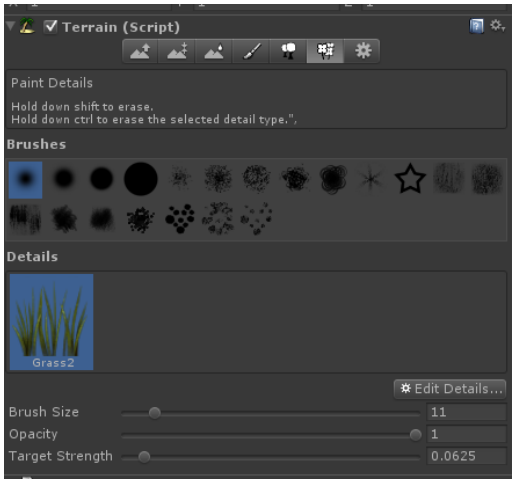


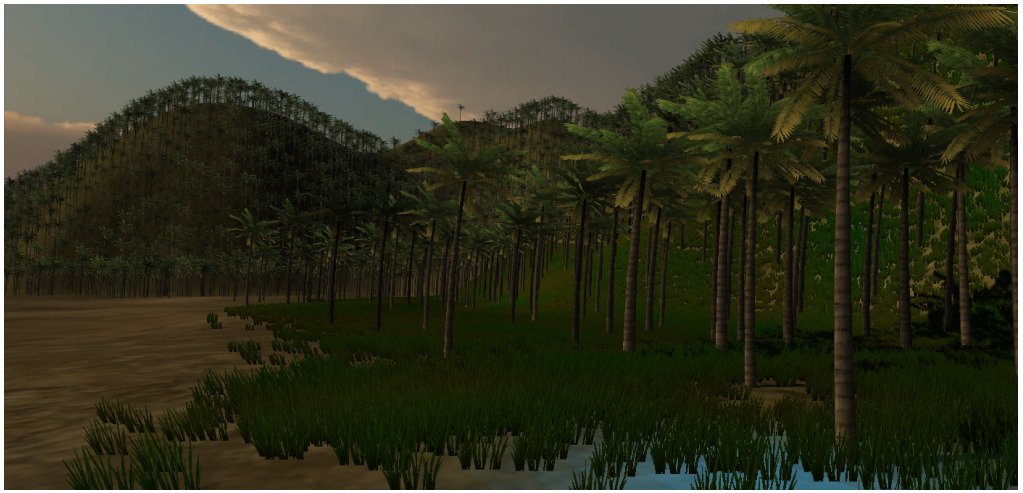


Les terrains

Pour ajouter de l'herbe, il suffit d'une texture d'herbe

- Au pinceau
- On peut faire varier la couleur, le placement
- Comme pour les textures : add et paint

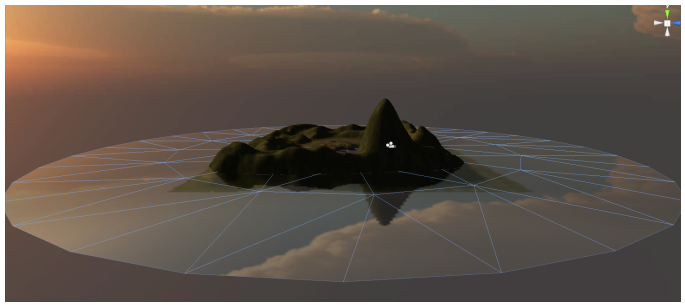




Etendue d'eau

Utiliser les objets de base Unity

- Shader spécial
- Utiliser la surface
- Modifier l'échelle pour correspondre au terrain



Le moteur physique

Le moteur physique

La physique dans Unity

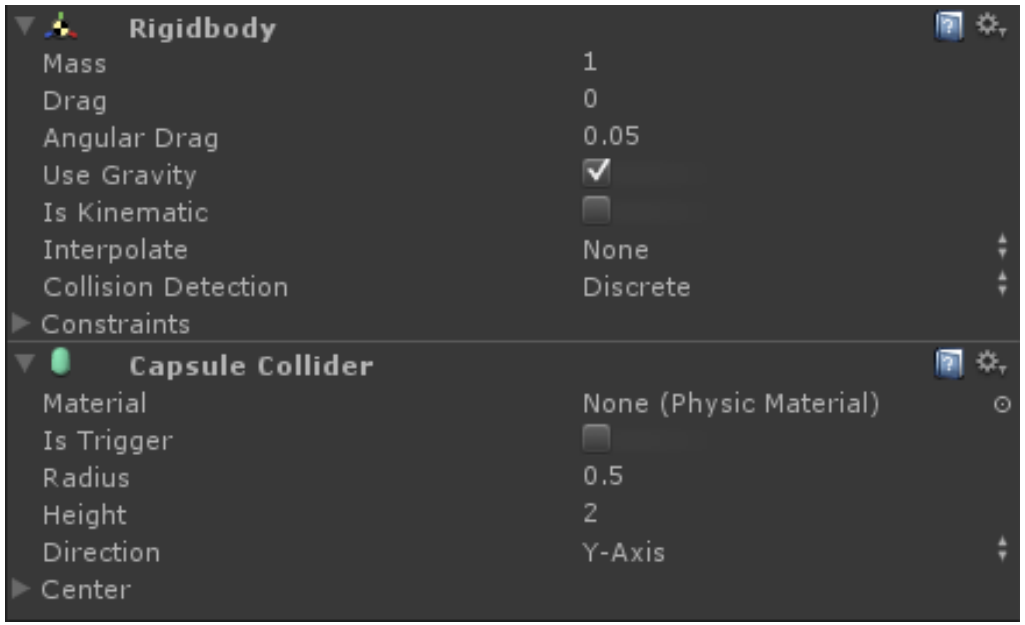
Unity utilise la librairie PhysX.
Permet d'appliquer les loi de la mécanique newtonienne.

- Gestion des collisions
- Gestion de la gravité
- Gestion des joints (contraintes entre objets)
- Transferts d'énergie (résultat d'une collision)

Les Rigidbodies

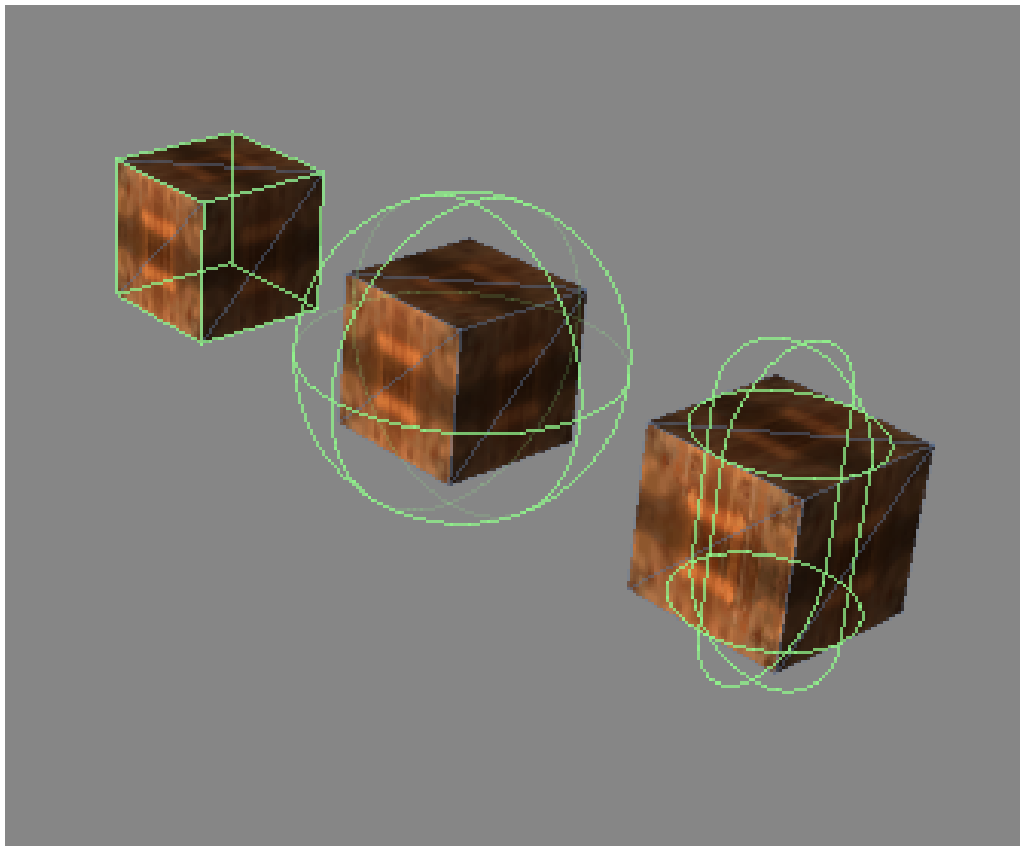
Pour qu'un objet soit soumis à la physique :

- Lui appliquer un RigidBody (c'est un component)
- Lui appliquer un Collider si il n'en a pas.



Les Rigidbodies

Plusieurs types de colliders (+ mesh et wheel) :



Manipuler un objet physique

On ne lui applique pas de transformation :

- On lui applique une force.
- Le moteur physique calcule sa nouvelle position.
- (Prise en compte de la masse, des collisions, etc...)
- Le moteur de rendu affiche la nouvelle position / orientation

```
1 //Applique une force de 10 vers les y positifs
2 rigidbody.AddForce (0, 10, 0);
```

IsKinematic

Si un objet est marqué IsKinematic

- Cet objet n'est pas soumis a la physique
- Ne se manipule pas avec des forces
- Par contre, il influence la physique du reste du monde (non Kinematic)

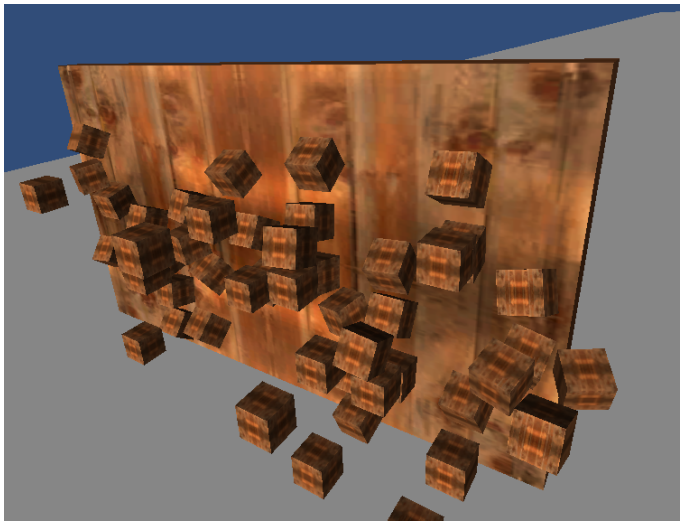
Très utile pour les objets animés :

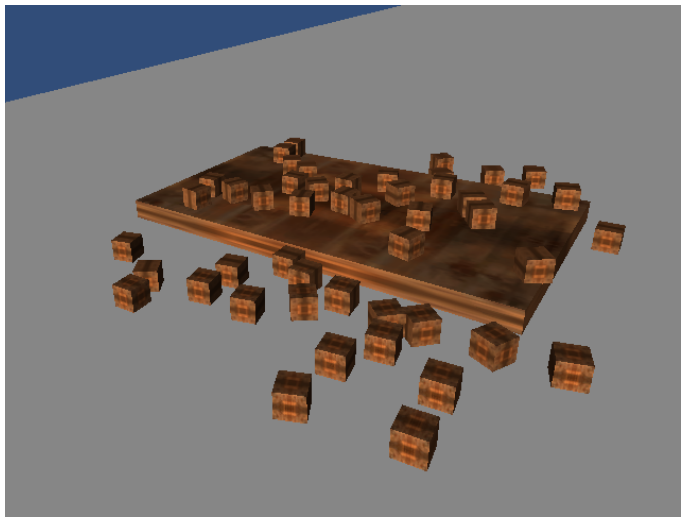
- Un ascenceur
- Une ragdoll en mode animé

Static colliders

Un static collider :

- A un objet collider
- N'a pas de RigidBody
- Il ne bougera pas, mais est pris en compte pour les collisions
- Très utile pour des murs par exemple
- Attention : ne pas le déplacer, ralentit le moteur.





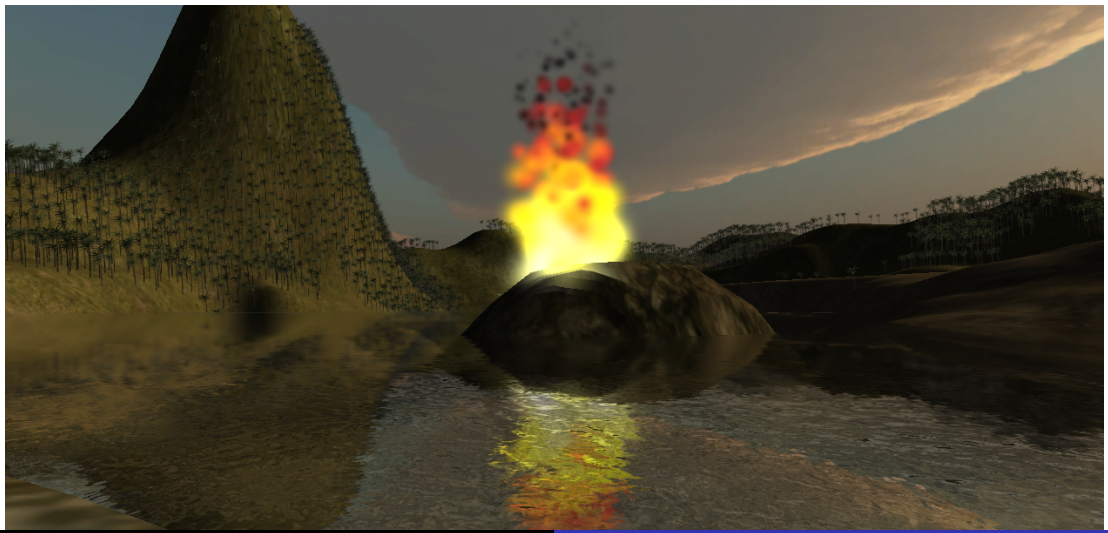
Système de particules

Système de particules

Système de particule

Un système de particules émet des particules :

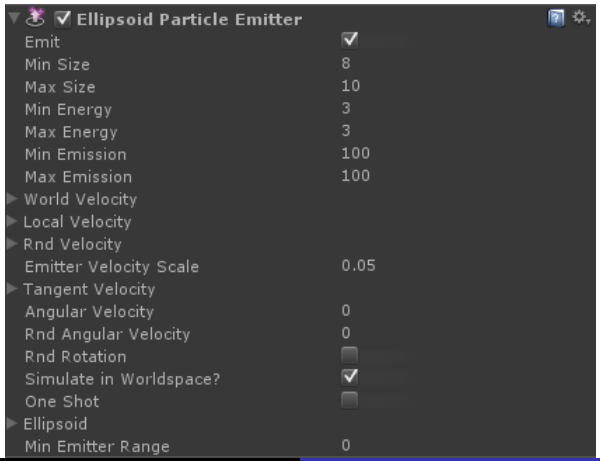
- Représentées par une simple texture
- Avec une vélocité initiale
- Une durée de vie programmée
- Soumises à diverses forces
- Animées tout au long de leur vie (couleur, rotation, taille)
- S'ajoute comme un game object (Particle System)



Le particule emitter

Gère les paramètres d'émission des particules (conditions initiales)

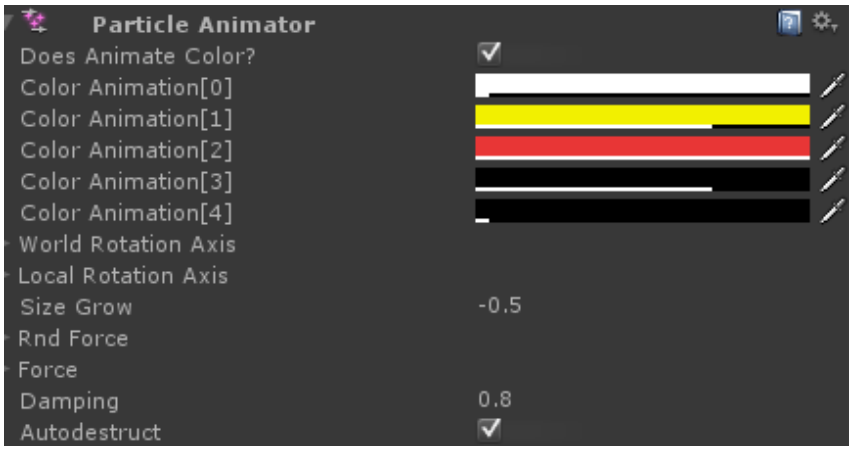
- Taille minimale et maximale
- Energie (durée de vie)
- Emission (nombre de particules par seconde)
- World / local velocity = vitesse d'émission
- Rnd Velocity = vitesse aléatoire ajoutée à la particule
- Emitter velocity scale : influence de la vitesse de l'emitter sur celles des particules



Le particule animator

Gère les particules tout au long de leur vie

- Transition entre couleurs tout au long de leur vie
- Rotation axis : les particules tournent autour
- Size grow : variation de taille
- Force : une force appliquée en permanence
- Damping : ralentissement des particules



Les animations

Les animations

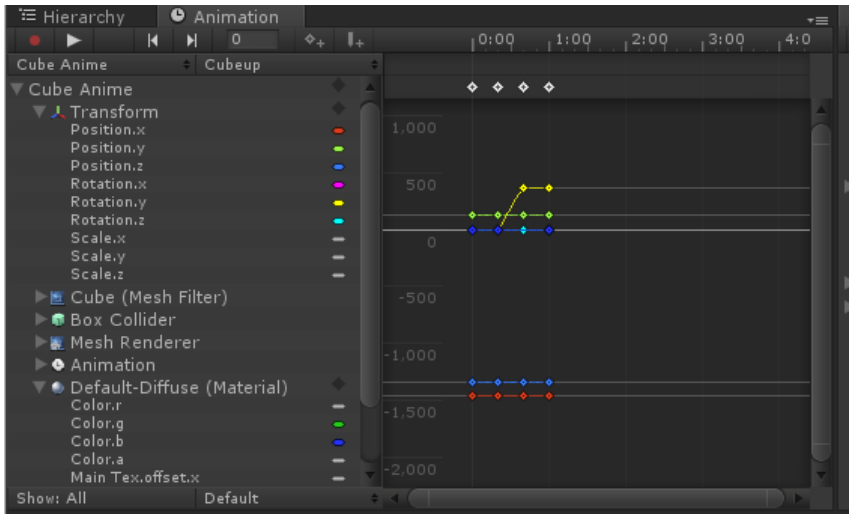
Les animations

- Peuvent être crées sur un logiciel externe
- Et ensuite importées
- Unity a son propre éditeur d'animations
- Suffisant dans de nombreux cas

Animation view

Se trouve dans "Windows->Animation"

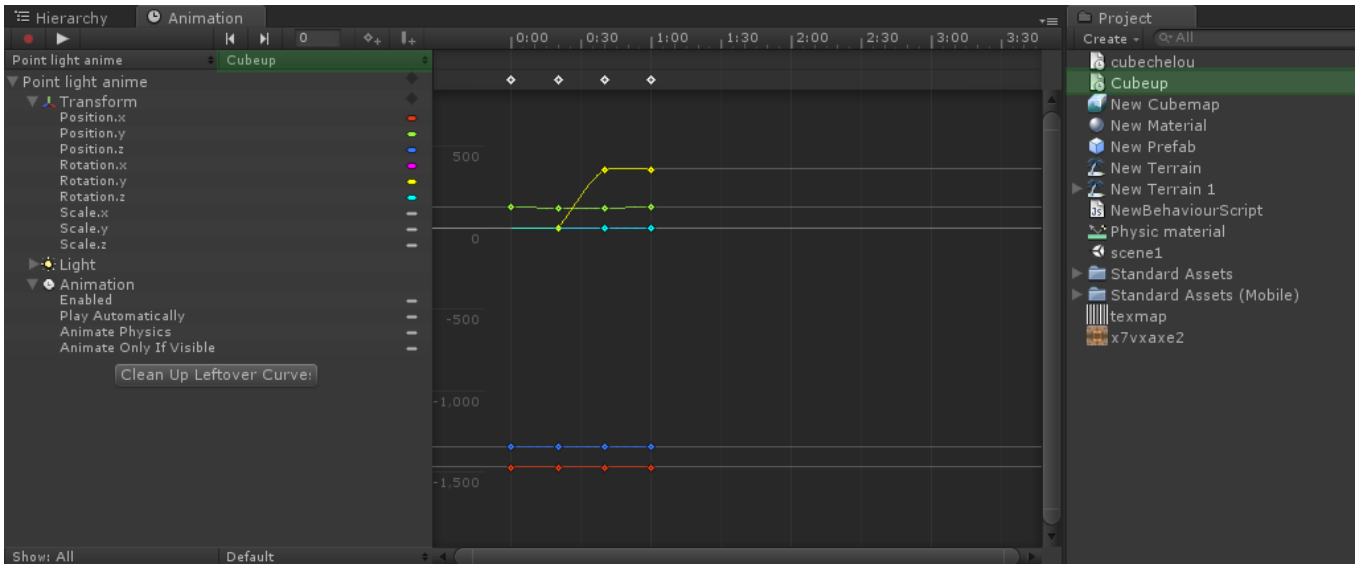
- Selectionner un GameObject (ici un cube par ex)
- Affiche tous les paramètres qu'on peut animer
- On place des key frames
- On édite les valeurs pour chaque keyframe
- On modifie les courbes de transitions entre keyframes



Animation view

Chaque animation est contenue dans un clip

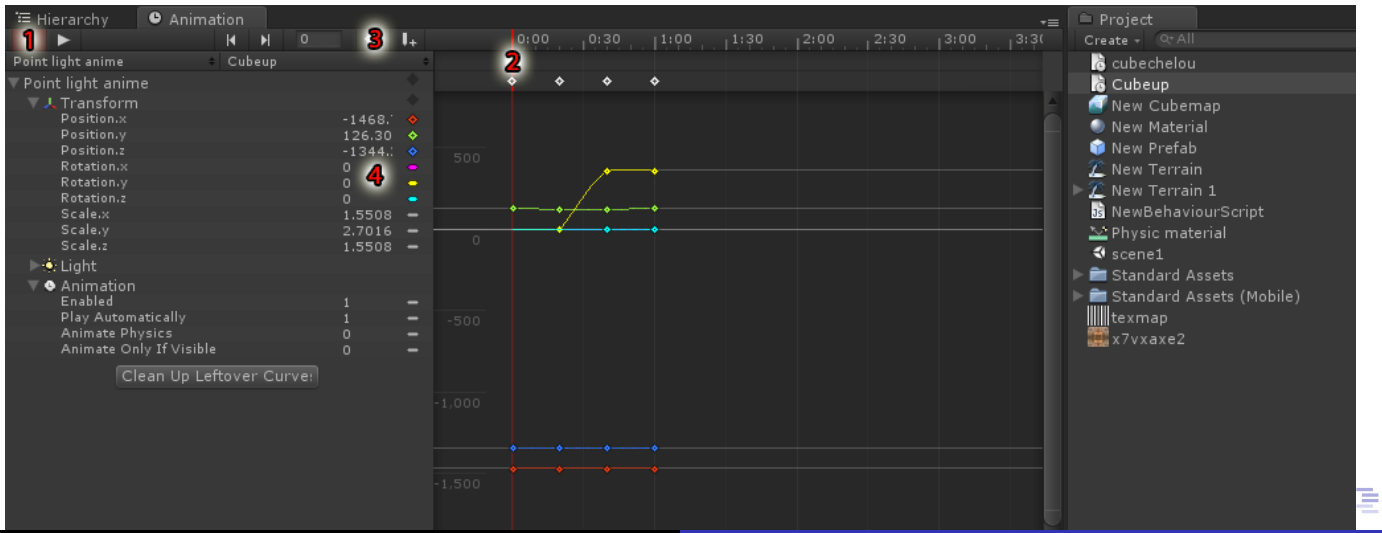
- Pour créer un clip : create new clip
- Chaque clip sera sauvé dans le répertoire d'assets
- On peut ensuite l'appliquer à un autre game object



Animation view

Pour animer un objet

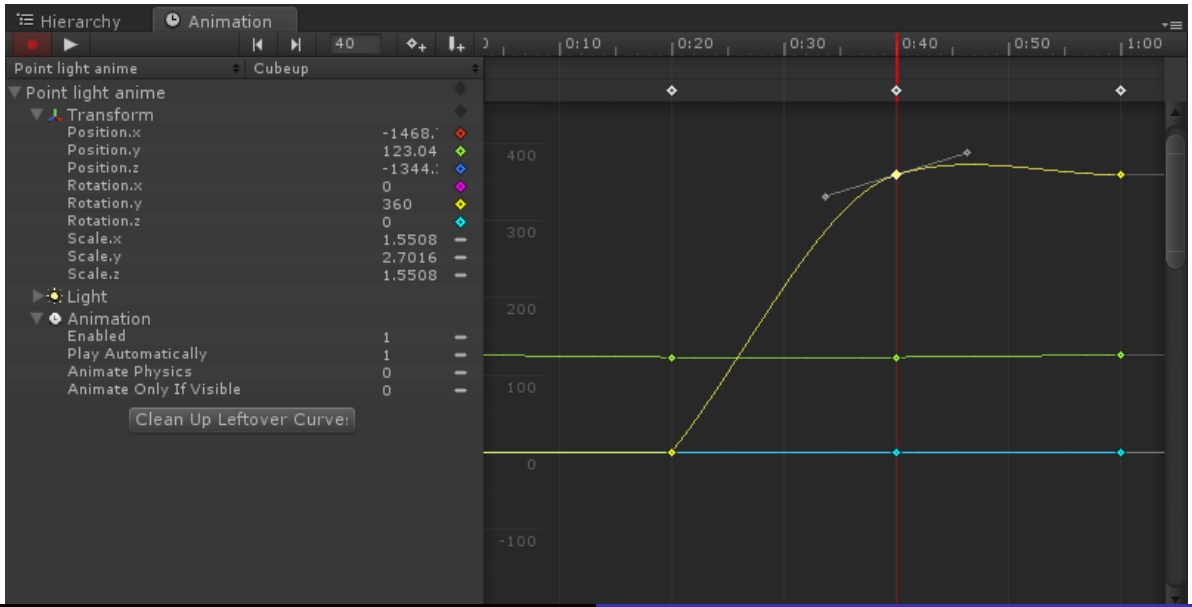
- 1) Passer en mode animation
- 2) Deplacer le marqueur rouge sur la timeline
- 3) Ajouter une étape clé (keyframe)
- 4) Modifier les paramètres (par valeur ou vue 3d)
- 5) Répéter 2-4 jusqu'à la fin
- 6) Quitter le mode animation (puis Play pour voir l'animation)



Animation view

Unity crée des transitions entre chaque keyframe :

- Ces transitions sont représentées par des courbes
- Ces courbes sont éditables
- On peut déplacer les keyframes
- Modifier les deux tangentes (forme de la courbe)
- Choix du mode de lecture en bas à gauche (Once loop, etc...)



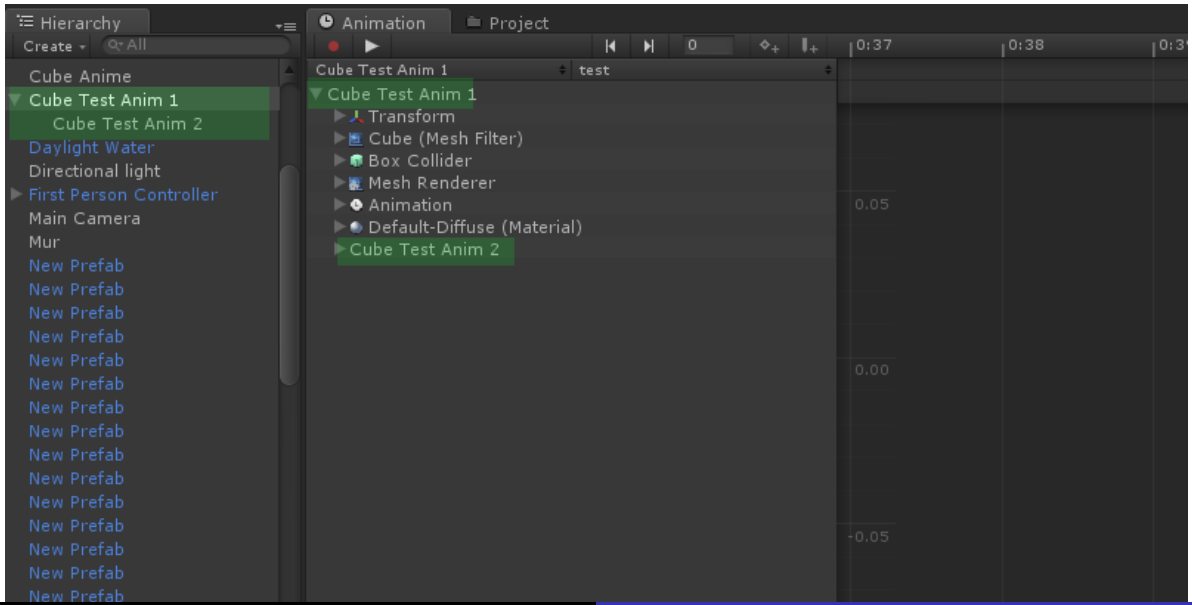
Animation view

Les animations sont hiérarchiques

- Une animation est en fonction du repère courant
- Un objet fils sera animé par rapport a la position du père

Pour une animation complexe

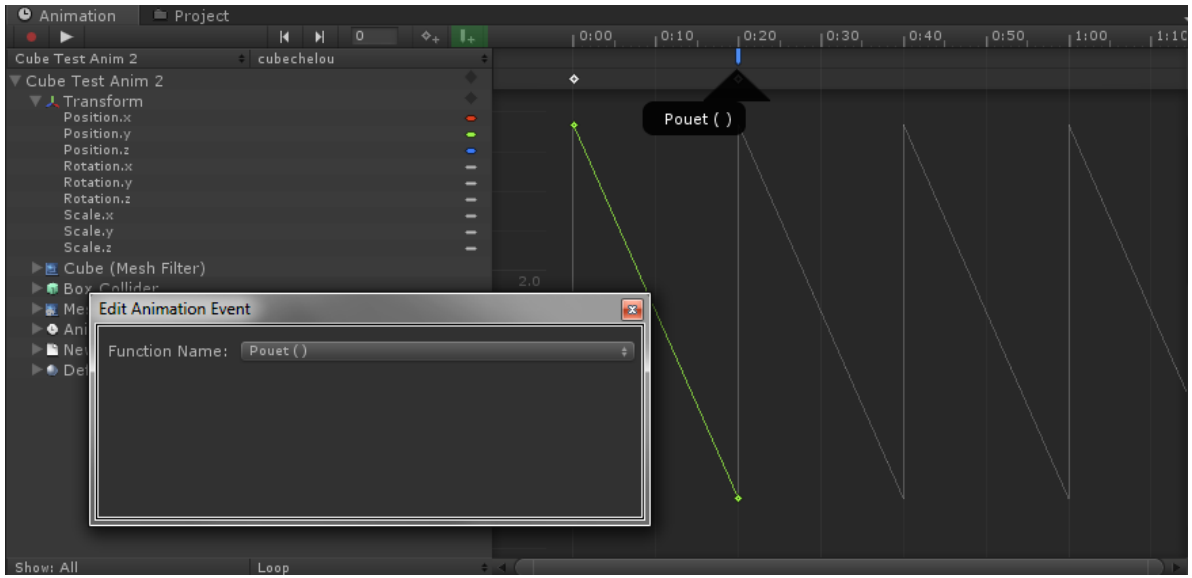
- Un père peut gérer l'animation de ses fils
- Placer l'animation sur l'objet le plus haut de la hiérarchie



Animation event

Une animation peut envoyer un évènement :

- A n'importe quel endroit de la timeline
- Il suffit d'ajouter un marqueur event
- Et de choisir une fonction déclarée dans l'objet



Creation d'un FPC

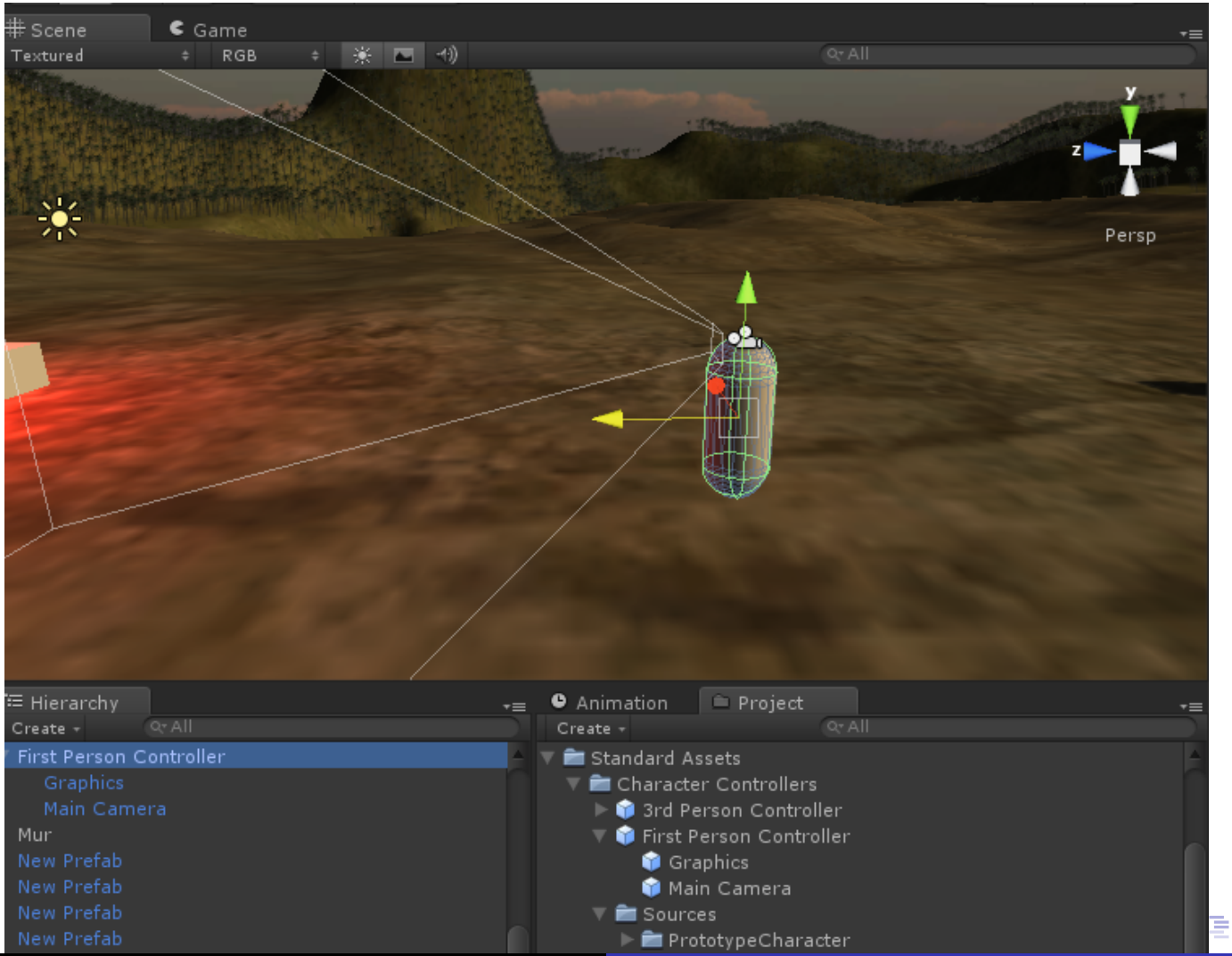
Creation d'un FPC (First Person Controller)

Le First Person Controller

Le FPC fait partie des assets standards.
Il permet de contrôler la camera au clavier et à la souris
Il se compose de trois objets :

- Un objet First Person Controller (Objet physique character controller)
- Un objet fils Graphics (une capsule : représente le joueur)
- Un objet fils Main Camera (la vue du joueur)
- Des scripts sont attaché au First Person Controller pour gérer lé déplacements.
- Pour l'utiliser, faire glisser dans la scène.

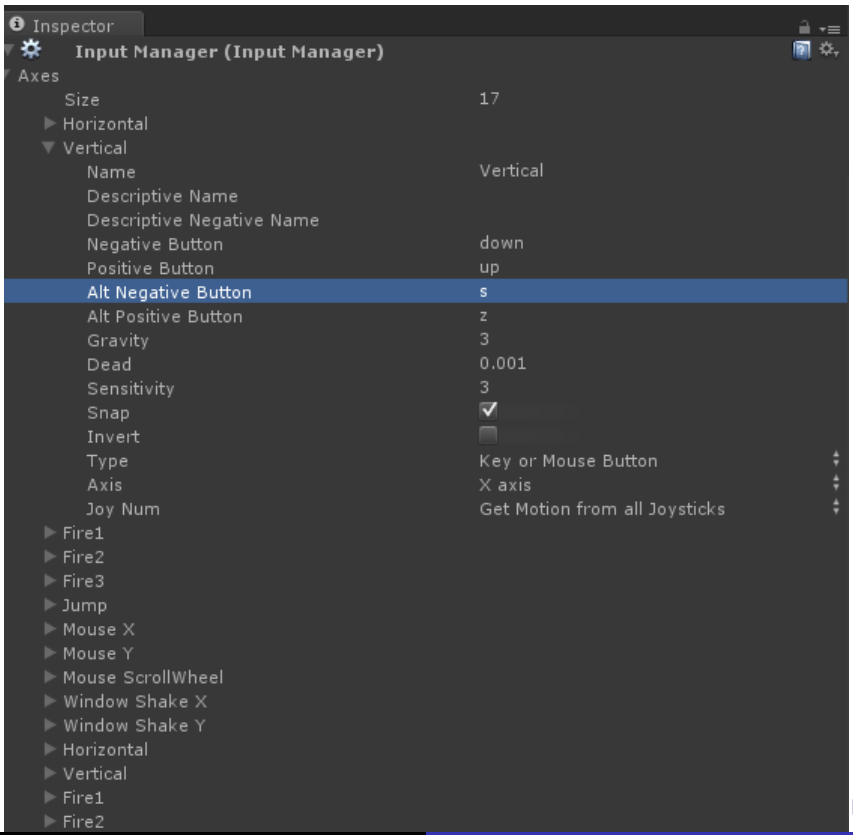
Le First Person Controller



L'Input Manager

Les scripts s'appuient sur une abstraction du clavier et de la souris.
On changer l'affectation des touches dans l'input manager :

- Edit -> Project Settings -> Input



Les scripts

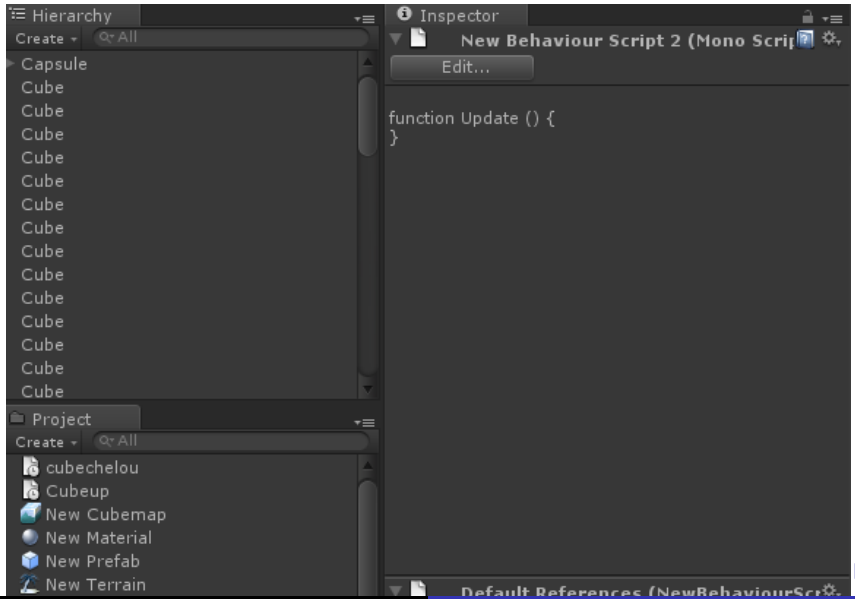
Les scripts

Les scripts

Les scripts sont des composants

Une fois attachés à un game object, ils font partie de son contexte.

- Project Window : Create -> [Choisir langage]
- Trois langages : Javascript, C# et Boo (python like).
- Cliquer sur le script
- Bouton Edit pour le modifier



Les langages

Les langages

Les scripts

Javascript :

- Très utilisés en Web
- Version modifiée pour Unity :
  - Pas de typages dynamique
  - Pas de prototypes
- Permet classes statiques
- serait 10 à 20 fois plus rapide que JS standard

Mon conseil : pratique pour codes à architecture simple, moins verbeux et plus permissif.

Les scripts

C# :

- Un des langages .NET
- Syntaxe proche de C++
- En gros même principe de classes et d'héritage
- Arithmétique de pointeurs dans un bloc *unsafe*
- Pas de delete : Garbage collector
- Typage fort
- Introspection
- Delegates

Mon conseil : Plus complet et restrictif, parfait pour une architecture complexe.

Les scripts

Boo :

- Teresa en japonais
- Fantome rond et blanc
- Aparait pour la première fois dans les chateaux de Super Mario Bros 3
- En vrai : je ne connais pas, spécifique mono et .NET

Mon conseil : apprendre les deux précédents.

Mono Develop

Mono Develop

Mono Develop

Bon outil de développement (mieux que l'éditeur de base)
Permet le debug
Dans Unity :

- Edit -> Preferences :
- Redéfinir l'éditeur de base pour monodevelop
- Verifier qu'on autorise bien le debug

Dans monodevelop :

- Tools -> Preferences -> Unity Debugger -> Editor Location.
- Redéfinir le debugger dans monodevelop

Mono Develop

Synchroniser les deux projets :
Dans Unity :

- Assets -> Sync MonoDevelop Project
- Permet de synchroniser unity et monodevelop
- Donne accès à l'ensemble des sources dous mono

Mono Develop

Pour pouvoir debugger :

- Passer en mode play sous unity
- Cliquer sur le bouton Attach de mono develop (une prise)
- Choisir le process unity
- Le debug est lancé
- Attention firewalls : connexion 127.0.0.1

Mono Develop

Monodevelop permet :

- Auto Completion
- Placer des break points
- Voir l'état de variables
- Voir la pile d'appels
- etc...

Scripting basique

Scripting basique

Les scripts

Deux évènements de base pour l'appel du code :

- **Update()** : appelé à chaque calcul de frame
- **FixedUpdate()** : appelé avant chaque mise à jour du moteur physique

Le code hors d'une fonction est appelé quand l'objet est chargé.

Les scripts

Attention ! faire des scripts temps réel :

- le temps écoulé entre chaque frame dépend de la vitesse de calcul
- toujours prendre en compte l'écoulement du temps
- **Time.deltaTime** est le temps écoulé depuis la dernière frame.

Les scripts

Une opération simple

```
1 void Update() {
2     //Rotation de 5 degres autour de l'axe des Y
3     transform.Rotate(0, 5, 0);
4 }
```

Les scripts

On peut accéder à divers composants de l'objet auquel le script est attaché :

Transform	transform
Rigidbody	rigidbody
Renderer	renderer
Camera	camera
Light	light
Animation	animation
Collider	collider

Sont accessibles par GetComponent<Type>()

```
1 transform.Translate(0, 1, 0);
2 GetComponent<Transform>().Translate(0, 1, 0);
```

Les scripts

Permet d'appeler une fonction d'un autre script du même objet

```
1 TheScript script = GetComponent<TheScript>();
2 script.DoSomething();
```

Les scripts

On peut également accéder à d'autres objets :
Objets fils de l'objet :

```
1 //Récupère le fils "Hand" et le translate
2 transform.Find("Hand").Translate(0, 1, 0);
3 //ou
4 //Translate tous les fils
5 Transform[] allChildren = GetComponentsInChildren<
      Transform>();
6 foreach (Transform child in allChildren) {
7     // do whatever with child transform here
8 }
```

Les scripts

Autres objets :

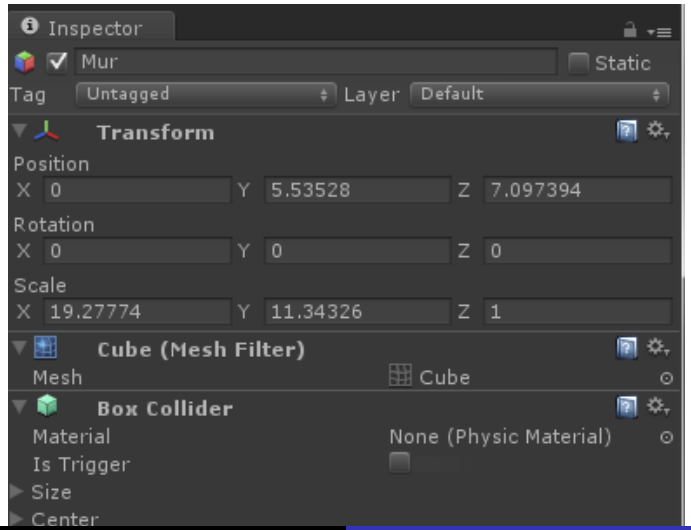
```
1 //Par nom
2 GameObject go = GameObject.Find("SomeGuy");
3 go.transform.Translate(0, 1, 0);
4
5 //Par tag
6 GameObject player = GameObject.FindWithTag("Player"
      );
7 player.transform.Translate(0, 1, 0);
```

Les scripts

Toute variable globale au script est accessible dans l'inspecteur :

```
1 public Vector3 aPosition = new Vector3(1, 1, 1);
2
3 [HideInInspector]
4 public float aHiddenFloat = 0f;
```

Si les variables sont du même type qu'un composant, elles sont affectables par drag and drop



Les scripts

Instantiation d'un objet :

```
1  Transform explosion;
2
3  // When a collision happens destroy ourselves
4  // and spawn an explosion prefab instead
5  void OnCollisionEnter (Collision collision) {
6    Destroy (gameObject);
7
8    Transform theClonedExplosion;
9    theClonedExplosion = GameObject.Instantiate(
         explosion,
10     transform.position, transform.rotation) as
           Transform;
11 }
```

Les scripts

Fonctions utiles :

Invoke	Invokes the method methodName in time seconds.
InvokeRepeating	Invokes the method methodName in time seconds.
CancelInvoke	Cancels all Invoke calls on this MonoBehaviour.
IsInvoking	Is any invoke on methodName pending ?
StartCoroutine	Starts a coroutine.
StopCoroutine	Stops all coroutines named methodName running on this behaviour.
StopAllCoroutines	Stops all coroutines running on this behaviour.

Les scripts

Fonctions redefinissables :

Update	Update is called every frame, if the MonoBehaviour is enabled.
LateUpdate	LateUpdate is called every frame, if the Behaviour is enabled.
FixedUpdate	This function is called every fixed framerate frame, if the MonoBehaviour is enabled.
Awake	Awake is called when the script instance is being loaded.
Start	Start is called just before any of the Update methods is called the first time.
Reset	Reset to default values.
OnCollisionEnter	Called when this collider/rigidbody has begun touching another rigidbody/collider.
OnCollisionExit	Called when this collider/rigidbody has stopped touching another rigidbody/collider.
OnCollisionStay	Called once per frame for every collider/rigidbody that is touching rigidbody/collider.

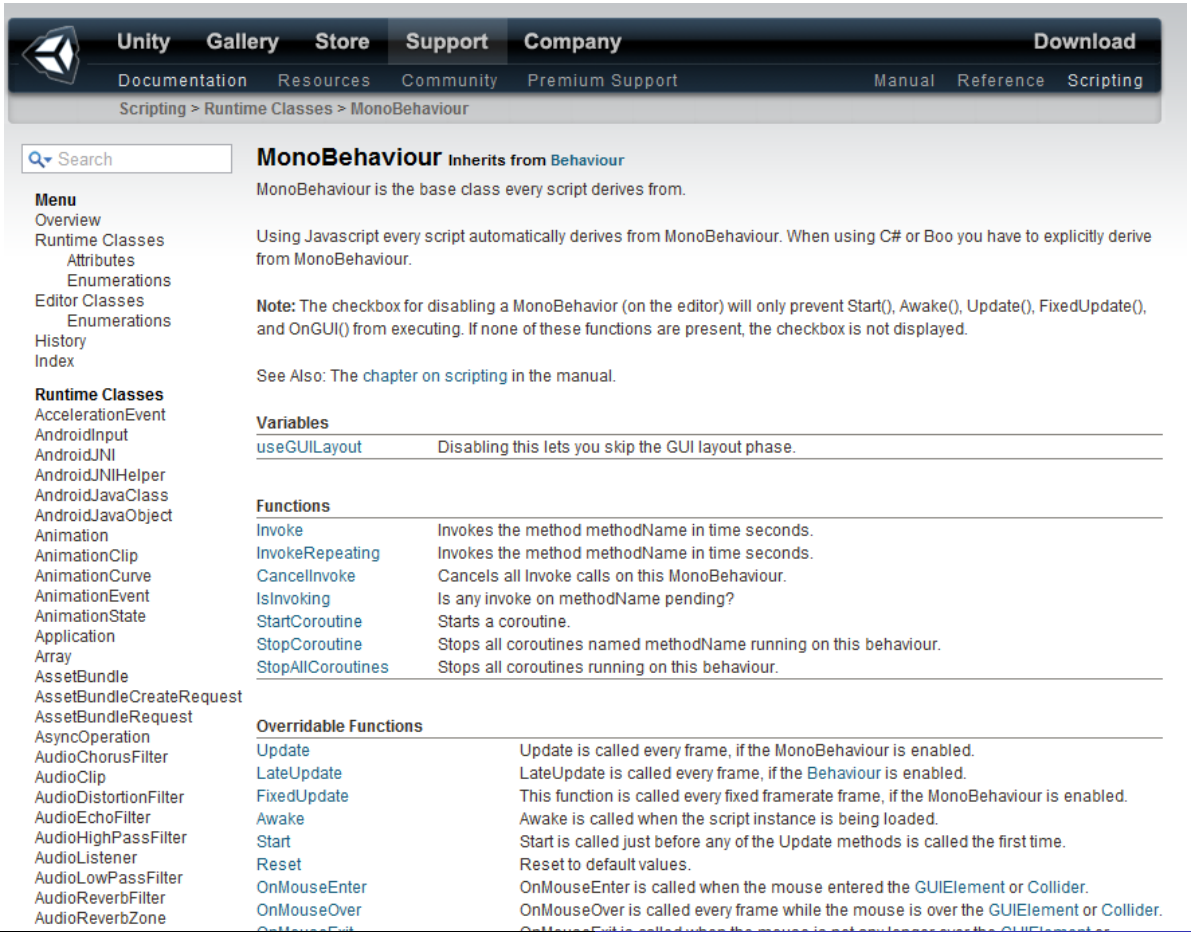
Les scripts

Envoyer un message :

```
1 // Calls the function ApplyDamage with a value of 5
2 SendMessage ("ApplyDamage", 5.0);
3
4 // Every script attached to the game object
5 // that has a ApplyDamage function will be called.
6 void ApplyDamage (float damage) {
7     Debug.Log (damage);
8 }
```

Les scripts

Toute la doc :
http ://unity3d.com/support/documentation/ScriptReference/MonoBehaviour.html



Les triggers

Les triggers

Les triggers

Un trigger :

- est un volume de collison non bloquant
- permet de déclencher un évènement quand le joueur est à un endroit donné
- mode de progression très utilisé dans le jeu vidéo
- narration spatiale, level design.

Plusieurs évènements :

- Entrée dans le trigger : OnTriggerEnter
- Sortie du Trigger : OnTriggerExit
- Dans le trigger : On Trigger Stay

Les triggers

Pour placer un trigger :

- Créer un gameobject de base
- Cocher, dans le composant collider, IsTrigger
- Permet de passer en mode non bloquant
- Attacher le code désiré sur les evts du trigger.

Les triggers

Les triggers sont des outils de LD :

- On doit les rendre invisibles dans la phase de jeu
- Utiliser un material semi transparent dans l'editeur
- Tips : utiliser emissive du Transparent/VertexLit
- Indépendant de la lumière.

Les triggers

```
1 Transform cube;
2
3 void Start(){
4   renderer.enabled = false;
5 }
```

Les triggers

```
1  void OnTriggerEnter(Collider other){
2    if(other.CompareTag("Player")){
3      for(int i = 0;i<10;i++){
4        for(int j = 0;j<10;j++){
5          Vector3 pos;
6          pos = transform.position;
7          pos += Vector3(i-5,20+Random.Range(0,5),j
               -5);
8          Instantiate(cube,pos, transform.rotation);
9        }
10     }
11   }
12 }
```

Class / Struct

```
1  class Test
2  {
3      public int A;
4  }
5
6  void Start()
7  {
8          Test test = new Test();
9      Test test2 = test;
10     test2.A = 10;
11     Debug.Log(test.A);
12     Debug.Log(test2.A);
13 }
```

Class / Struct

Reference sur meme objet

```
1 | 10
2 | 10
```

Class / Struct

```
1  struct Test
2  {
3      public int A;
4  }
5
6  void Start()
7  {
8          Test test = new Test();
9      Test test2 = test;
10     test2.A = 10;
11     Debug.Log(test.A);
12     Debug.Log(test2.A);
13 }
```

Class / Struct

Duplication de la structure

```
1 | 0
2 | 10
```