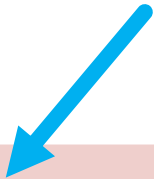


Introduction aux shaders

Les étapes du jeux

Fonction update code C# unity



Application

- CPU
- Input
- Code du jeu (Collision, mecanique, ect.)

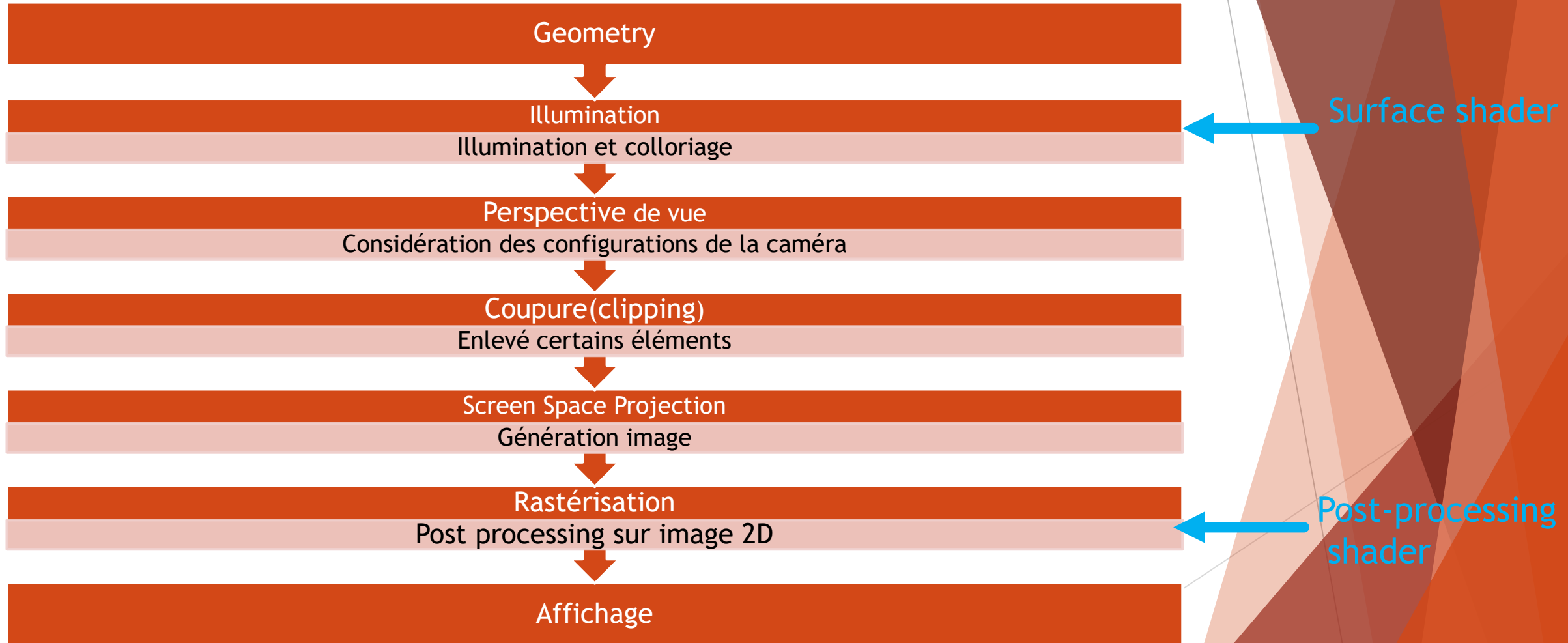
Geometry

- Position des objets et transformation et des polygones

Rastérisation

- Produit l'image

Contenus de Geometry + Rastérisation



Qu'est-ce qu'un shader ?

Historiquement utiliser pour générer l'ombrage, les shaders sont aujourd'hui utilisés pour générer plusieurs effets visuels et donner un rendu à des objets d'une scène.

Que fait un shader ?

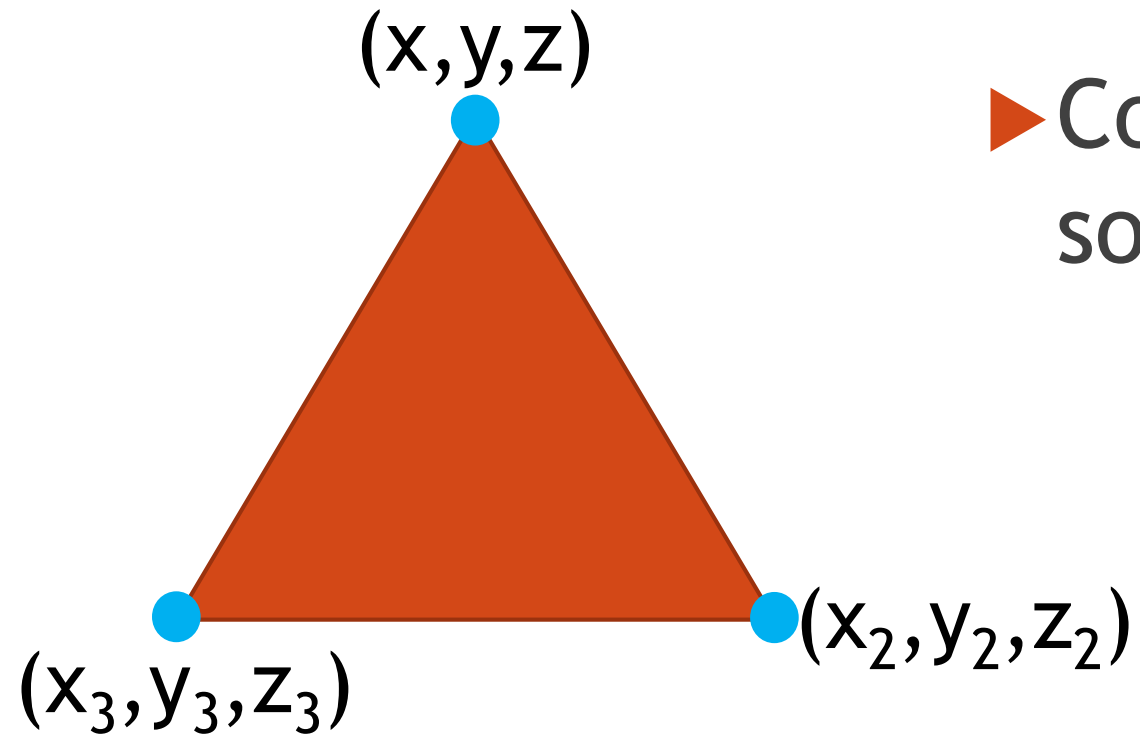
- ▶ Il peut définir ou modifier la couleur d'un pixel
 - ▶ Saturation, teinte, luminosité, contraste
- ▶ Modifier la transparence
- ▶ Modifier les sommets des primitives

Pour définir une forme en 3D

Pour définir une forme en 3D, il est nécessaire d'avoir un « Polygon Mesh ». Celui-ci est composé de :

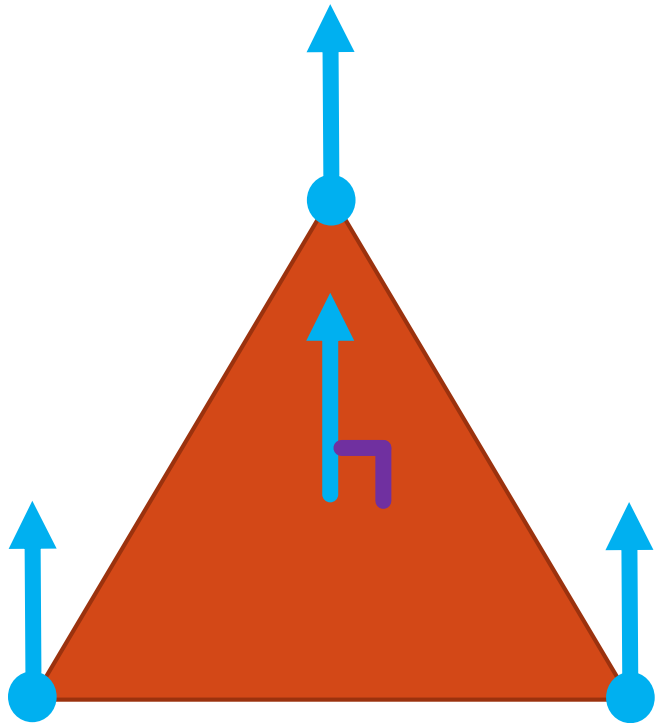
- ▶ Des sommets ou vertices
- ▶ Des arêtes ou edges
- ▶ Des faces

Conservation d'un polygone(1)



► Conservation des sommets (vertices)

Conservation d'un polygone(2)

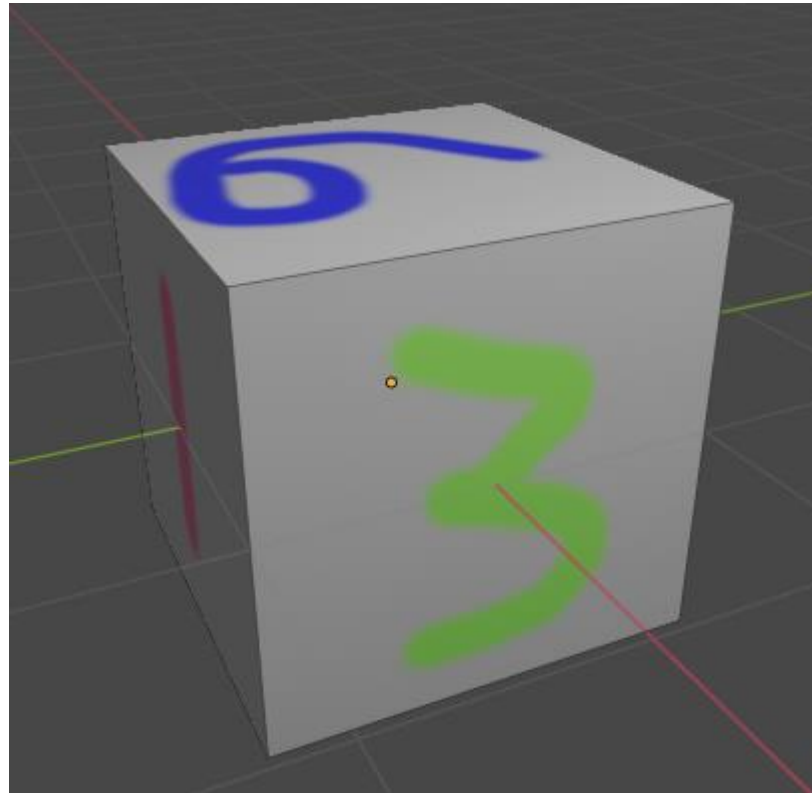
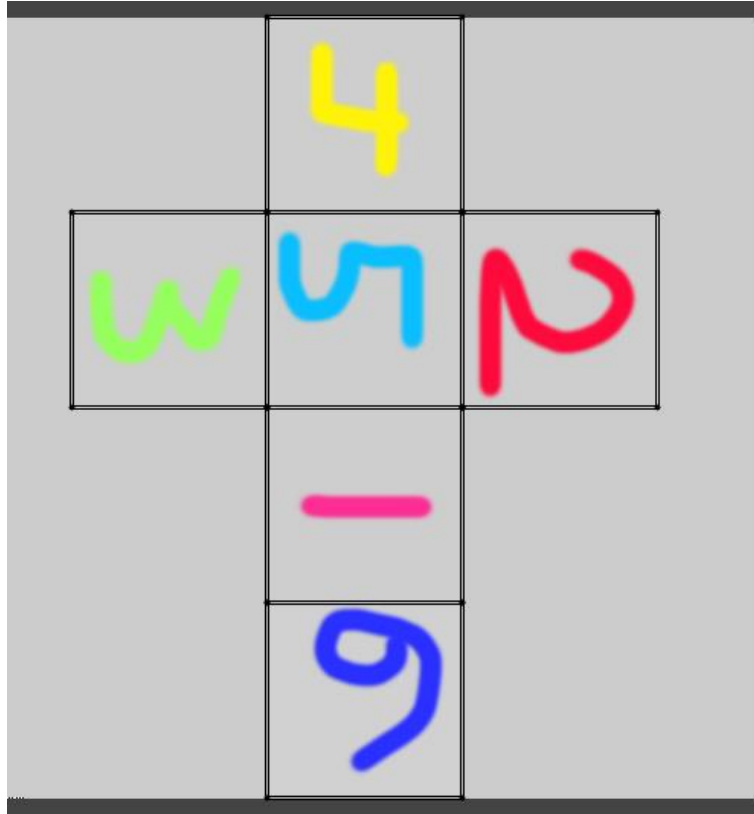


- Conservation des normales des faces et des sommets

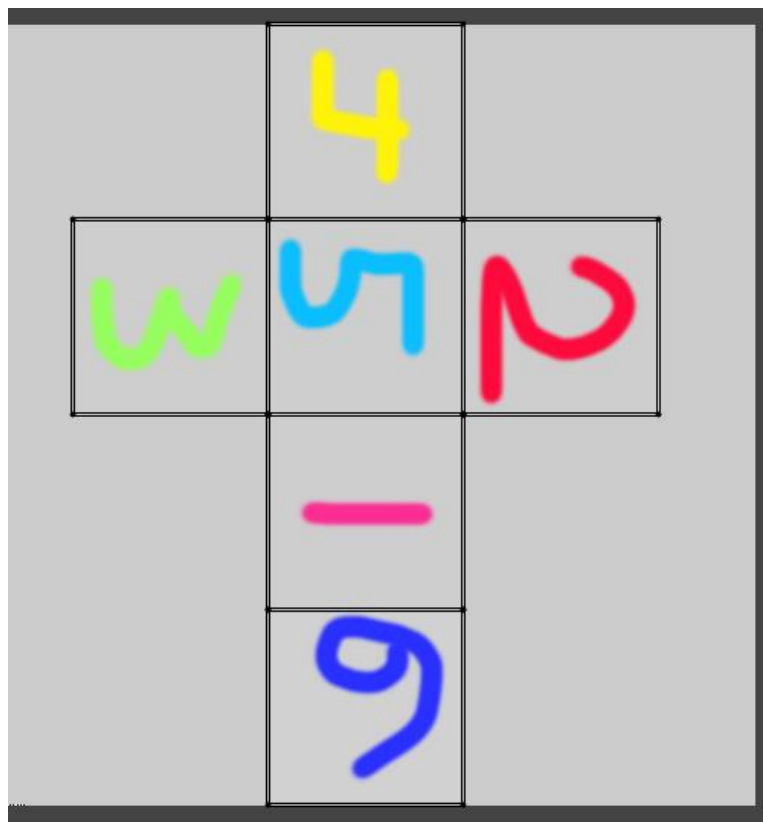
Les UVs

Indique les coordonnées pour comprendre comment la texture est « wrap » sur l'objet. Il faut le voir comme du papier d'emballage. Pour certaines formes c'est évident alors que pour d'autres l'emballage est plus compliqué. Cette technique permet la continuité d'une texture sur plusieurs faces (polygones) sans « seams » (démarcation de couture).

Exemple de UV dans blender

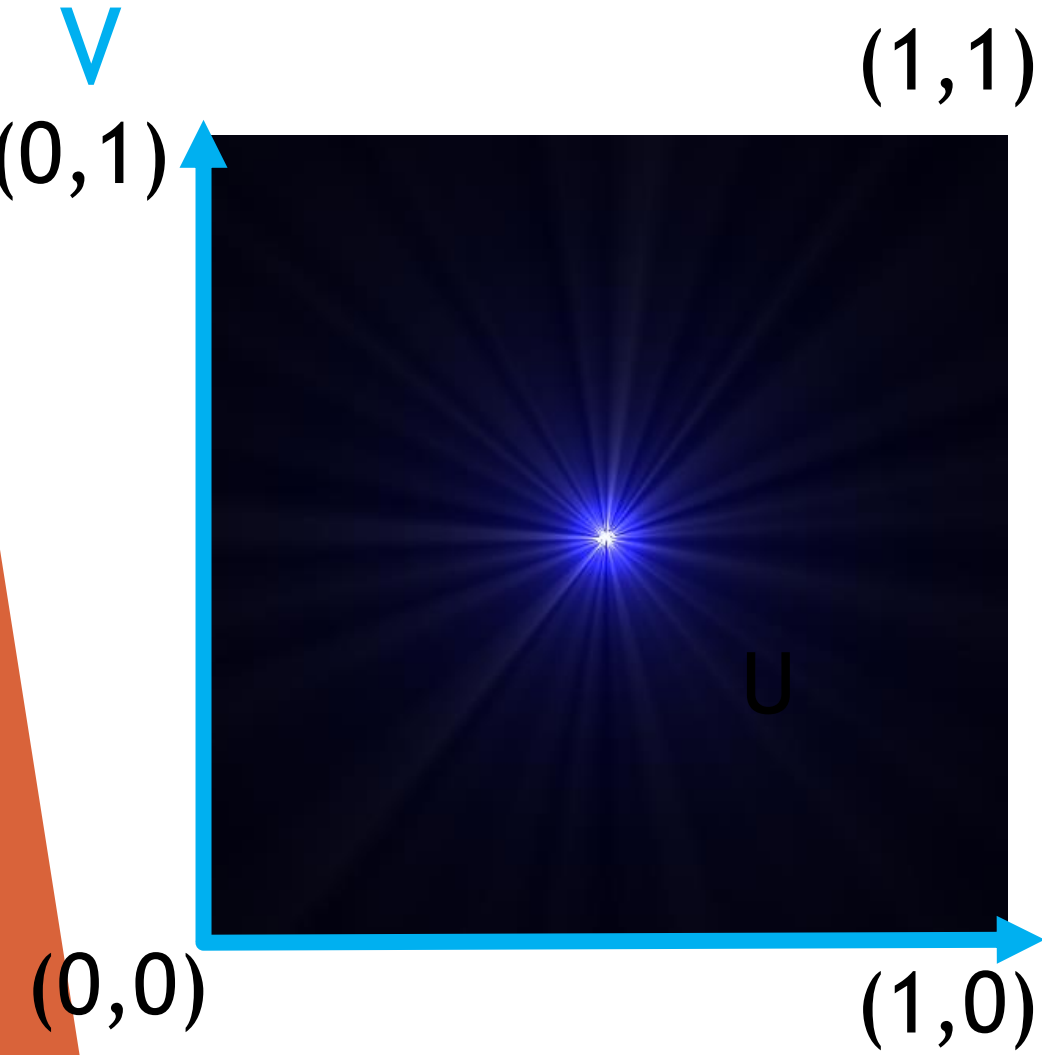


Exemple de UV dans blender



- ▶ Chaque région représente une face de notre mesh.
- ▶ Chaque sommet est donc associé à une coordonnée dans la texture

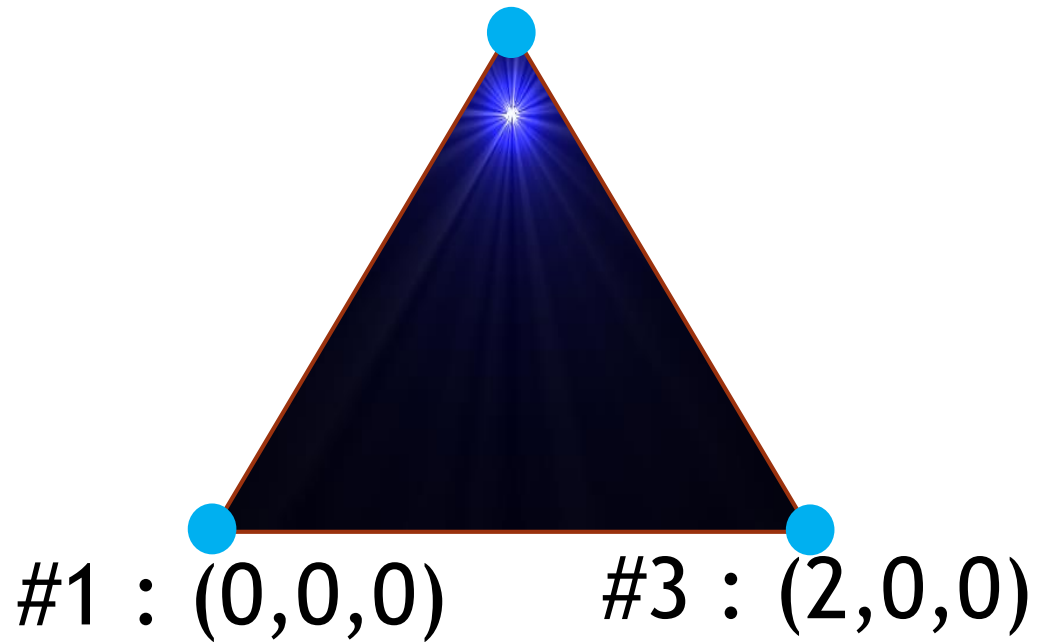
Coordonnée UV



Coordonnée entre U et V seront entre 0 et 1 et ce peut importe la taille de la texture. Il faut le voir comme un pourcentage et pas comme une position exacte.

Coordonnée de UV dans un shader

#2 : (1,2,0)



UV Array

1 : (0,0)

2 : (0.5,0.6)

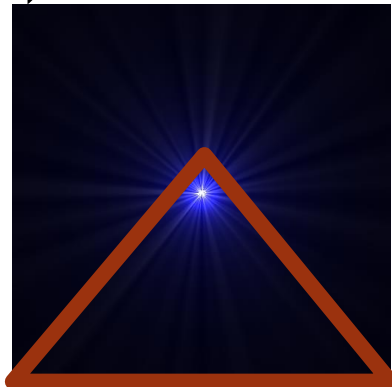
3 : (1,0)

Vertex Array

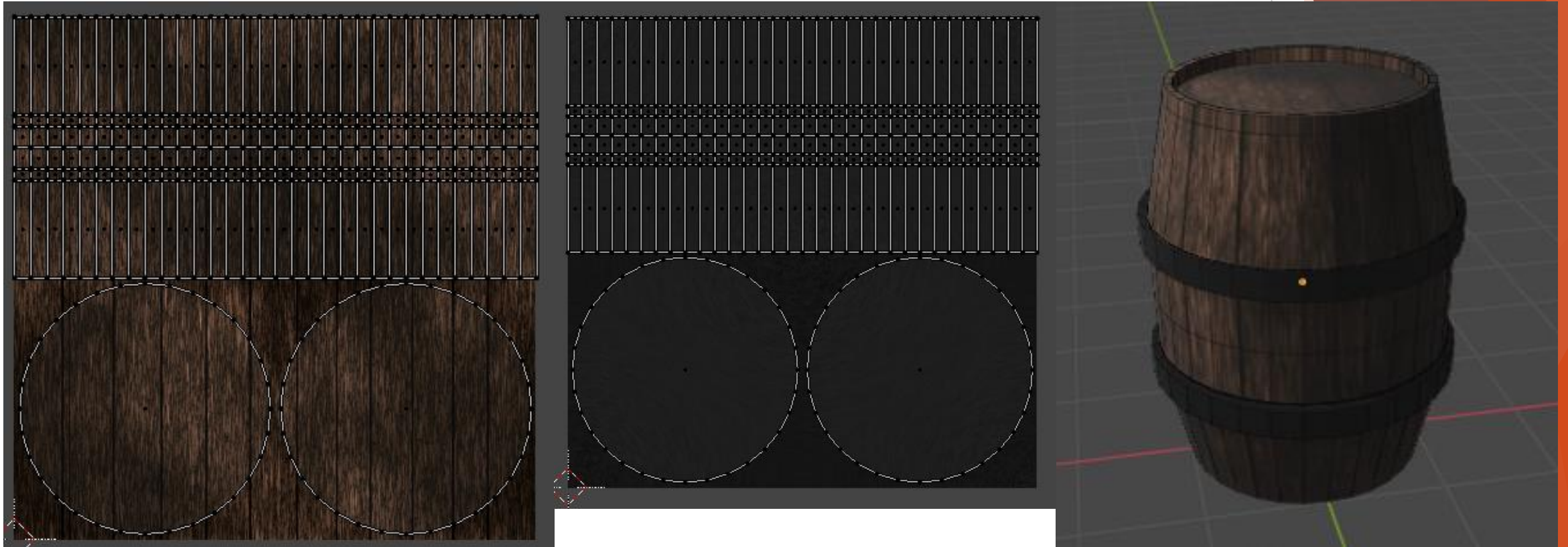
1 : (0,0,0)

2 : (1,2,0)

3 : (2,0,0)



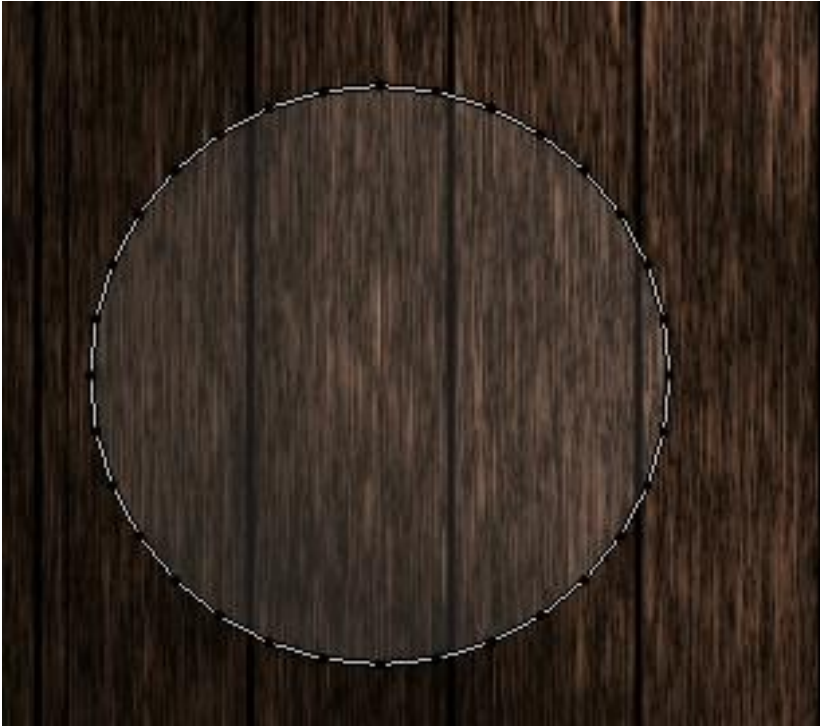
Exemple2 de UV dans blender



Exemple3 modification scale UV(avant)



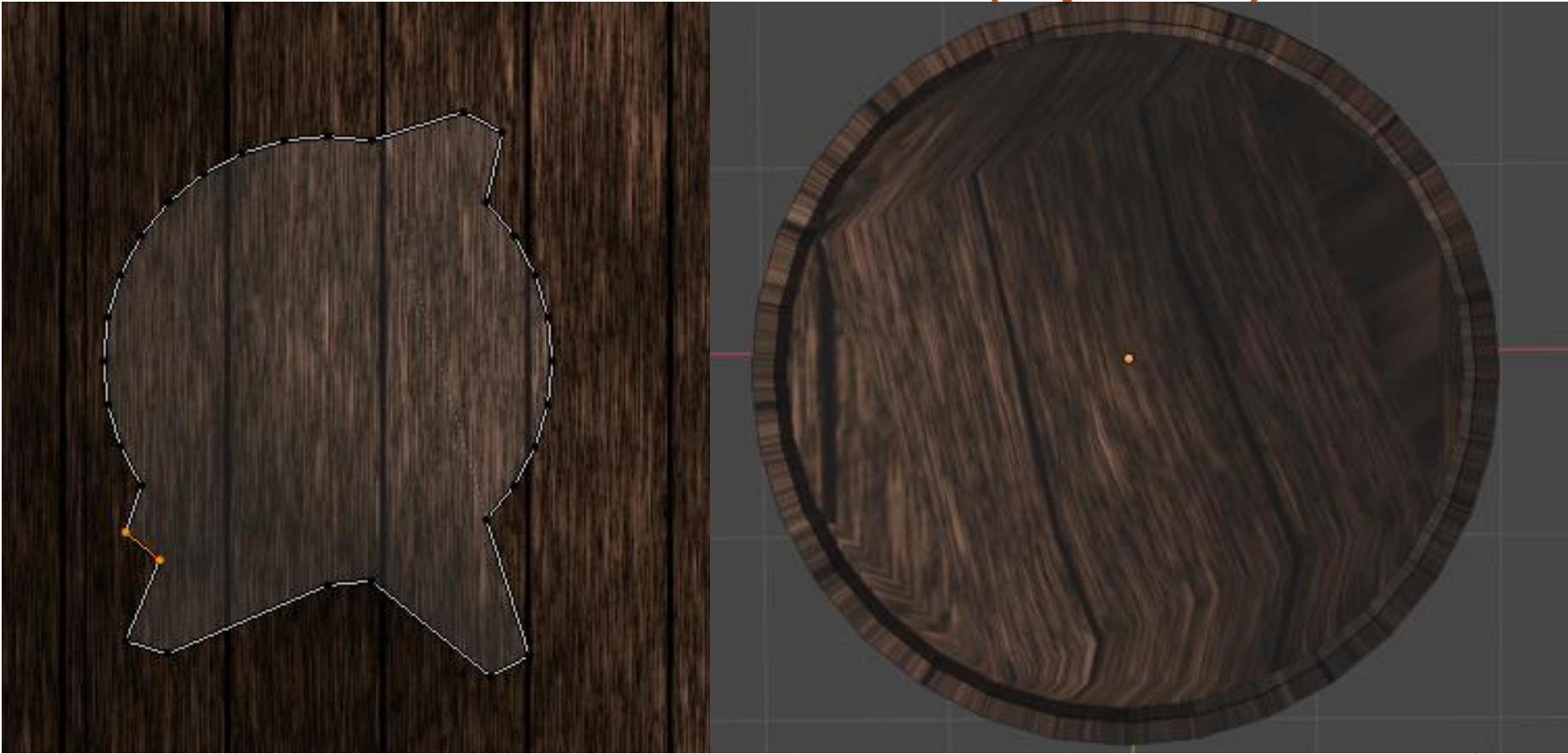
Exemple3 modification scale UV(après)



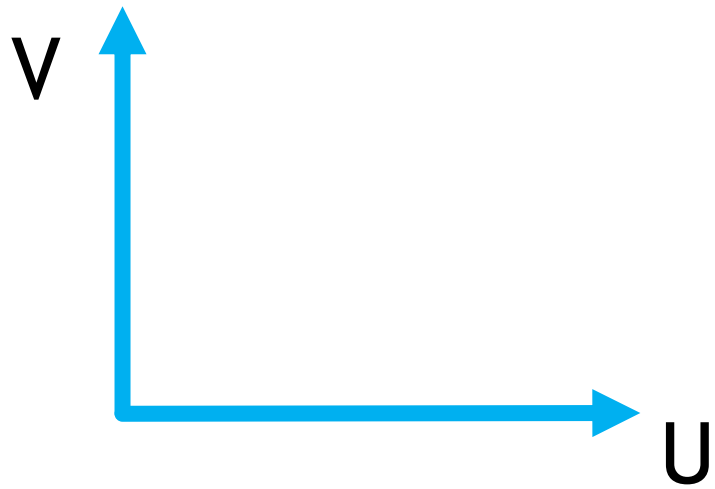
Exemple3 modification scale UV(après)



Exemple4 déformation modification scale UV(après)

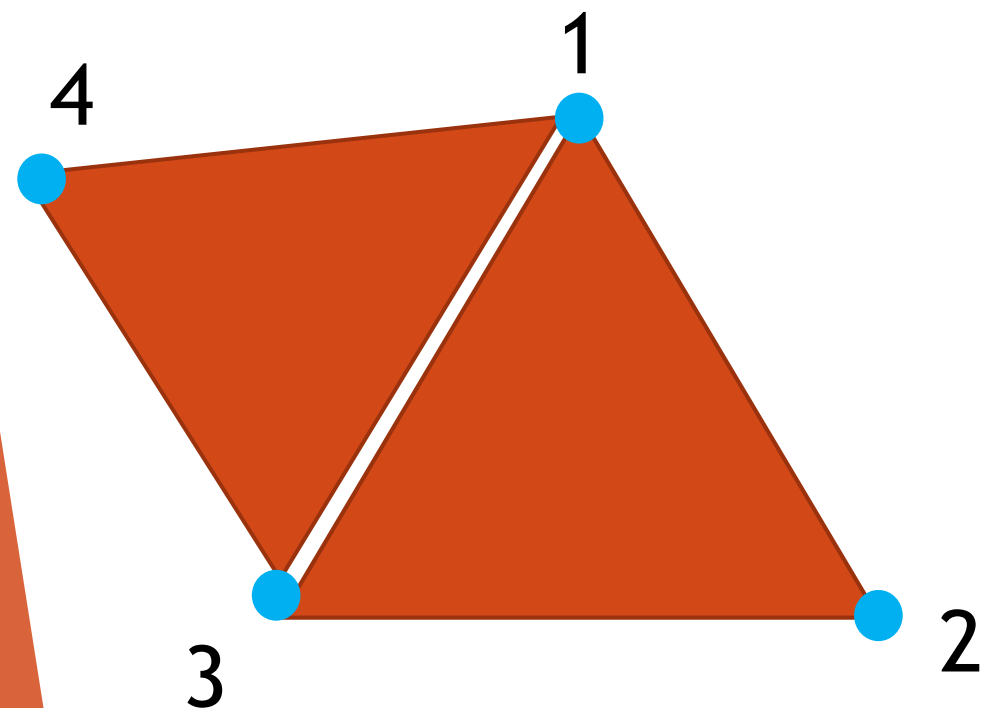


Résumé UV



Un UV représente un point d'une texture qui est associé à un point sur un polygone. U représente les X et V les y.

Conservation d'un polygone(3)



- Conservation des numéros de sommets qui ensemble forment un polygone

PolygoneArray

1,2,3

1,3,4

Un mesh contient donc

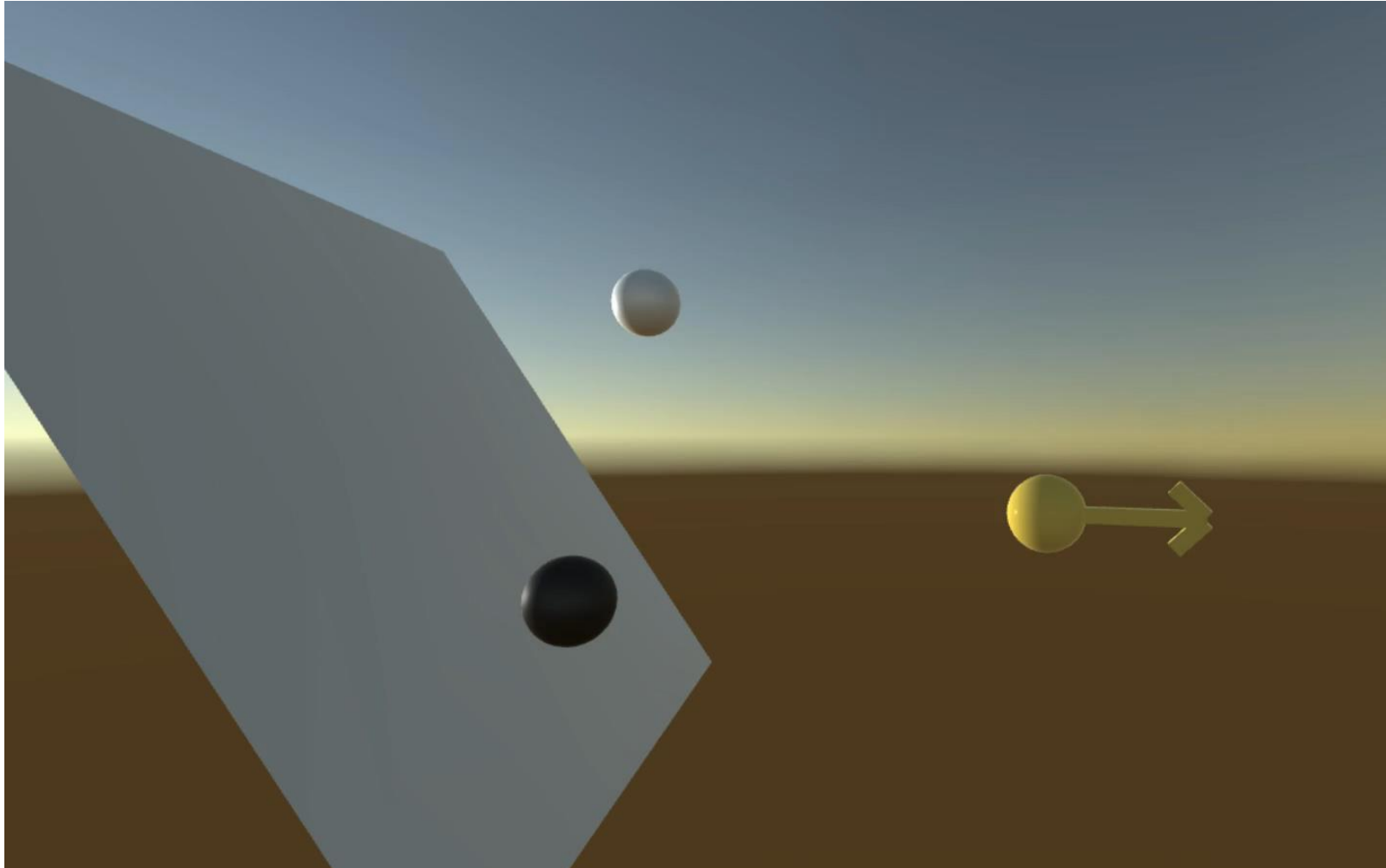
- ▶ Un tableau des vertex/sommet
- ▶ Un tableau des normales
- ▶ Un tableau des UV
- ▶ Un array des polygones

L'illumination

Alex Gagnon

22

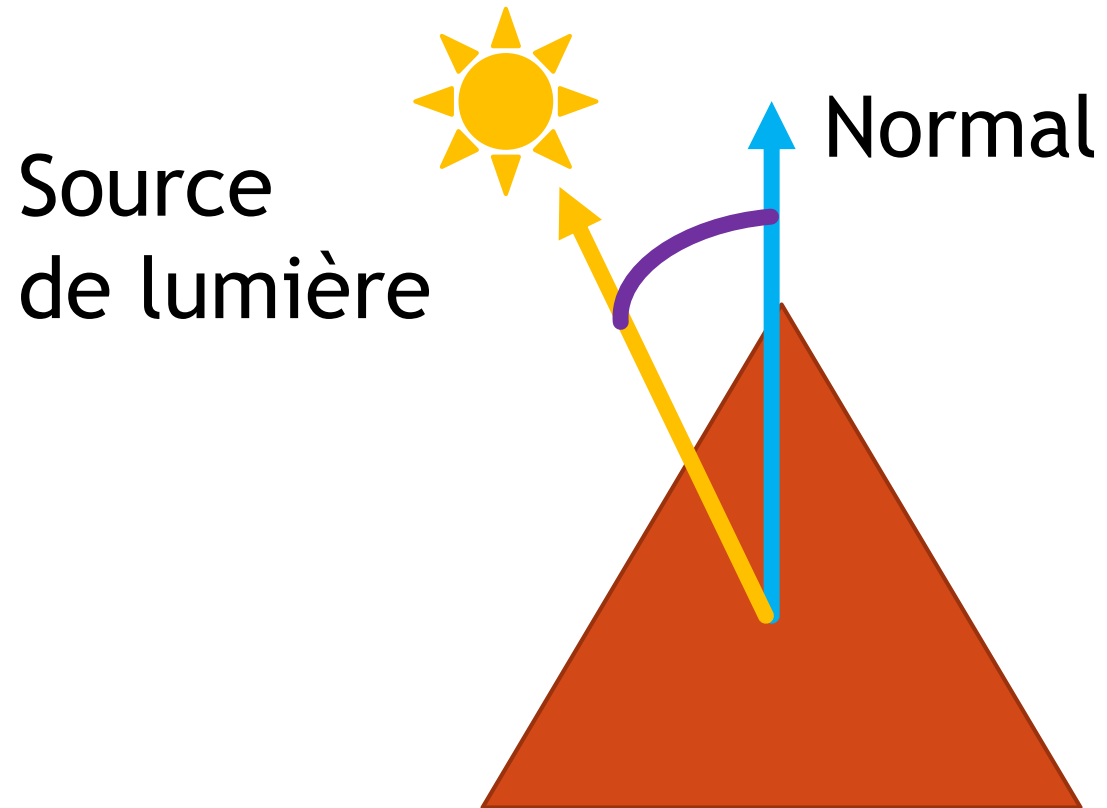
Illumination



Le modèle le plus simple est le modèle Lambert

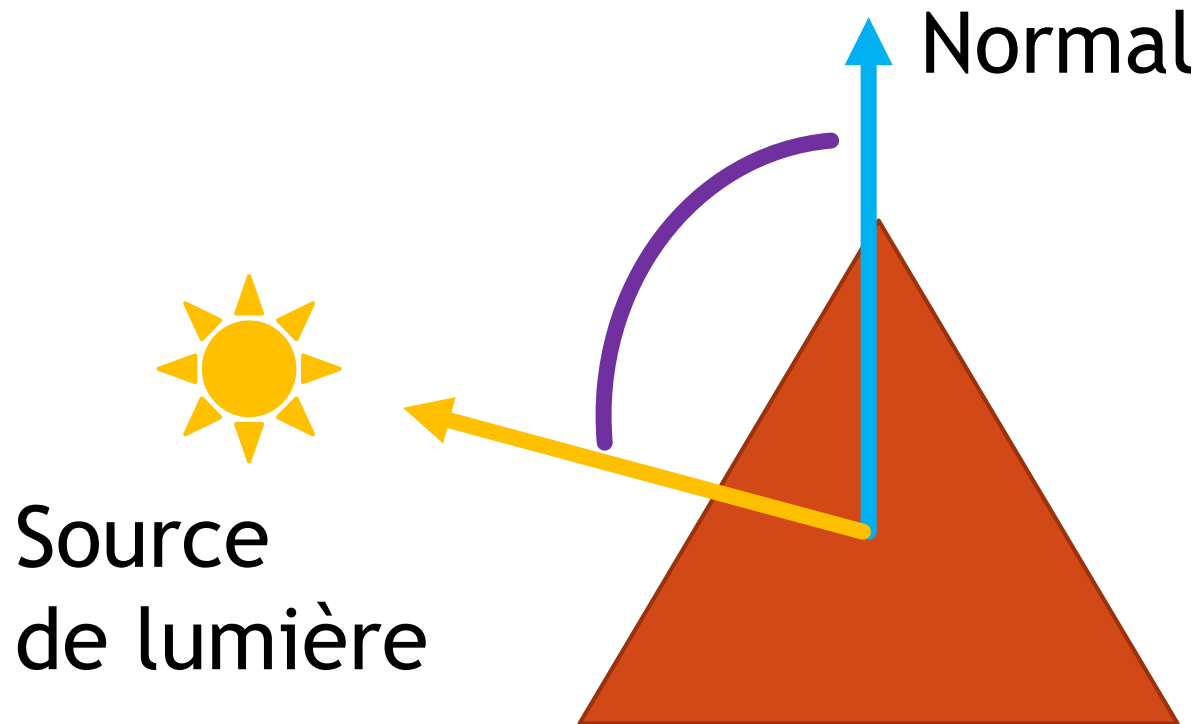
Ce modèle permet de déterminer la luminosité d'un endroit sur un objet. Ce modèle fut inventé par Johann Heinrich Lambert et utilise la source de lumière avec la normale d'une surface.

Composantes nécessaires pour calculer la lumière(lambert)



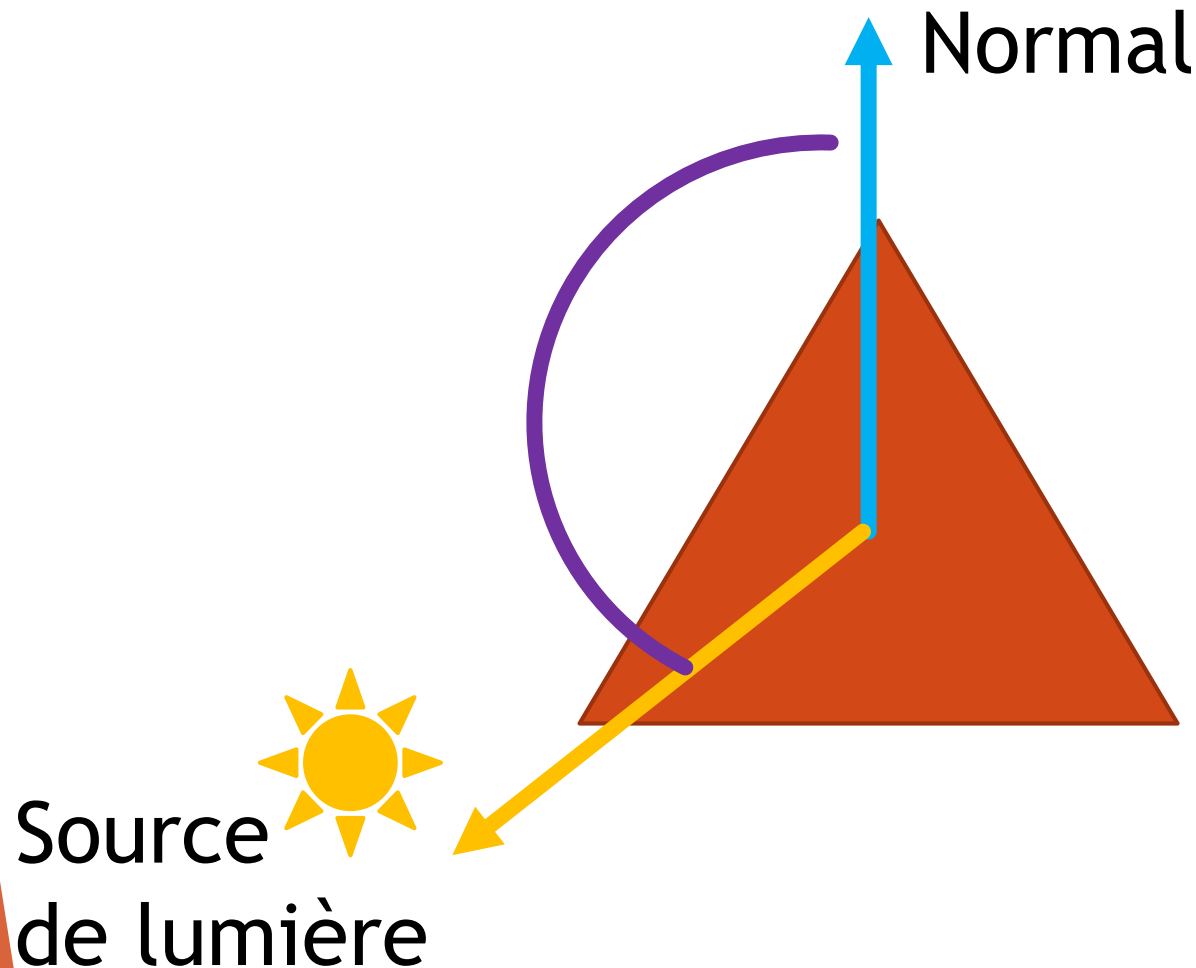
...mais un petit angle indique une grande luminosité.

Composantes nécessaires pour calculer la lumière(lambert)



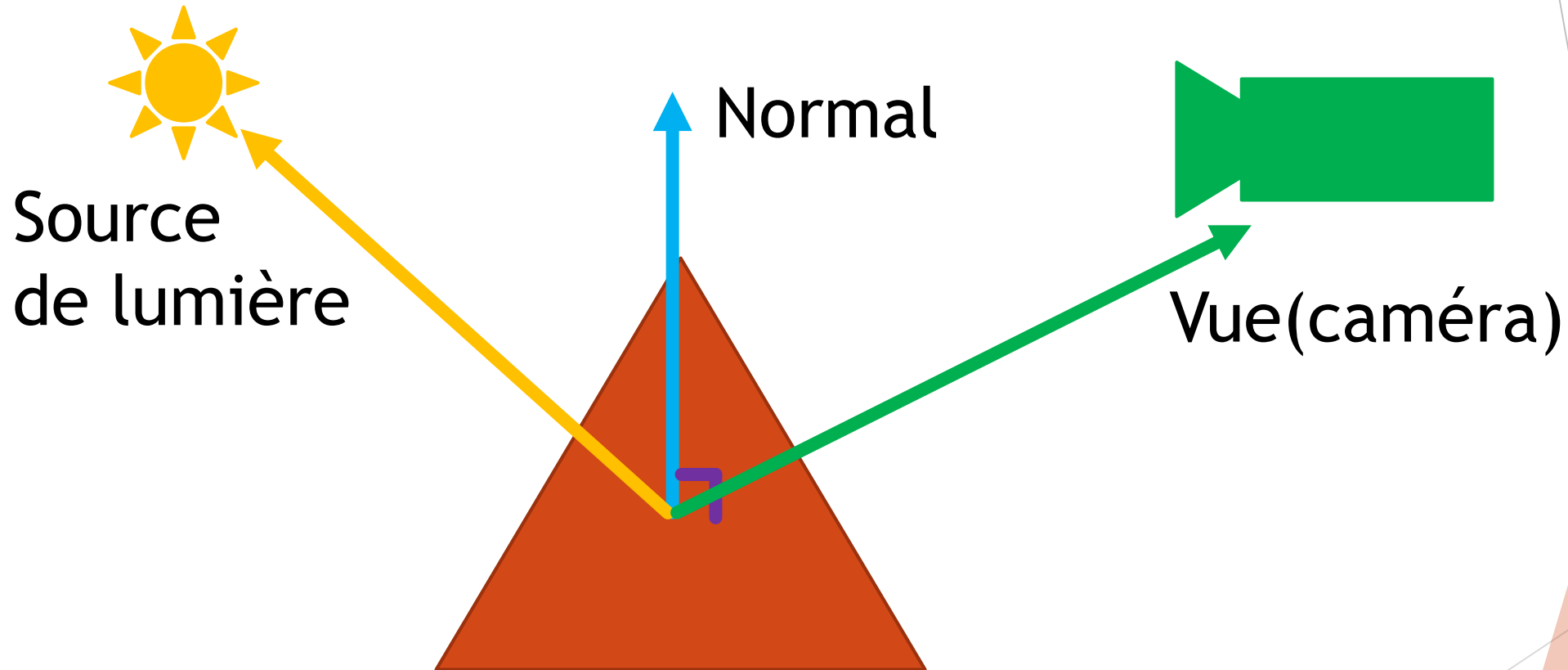
Dans ce modèle, un grand-angle indique peu de luminosité.

Composantes nécessaires pour calculer la lumière(lambert)



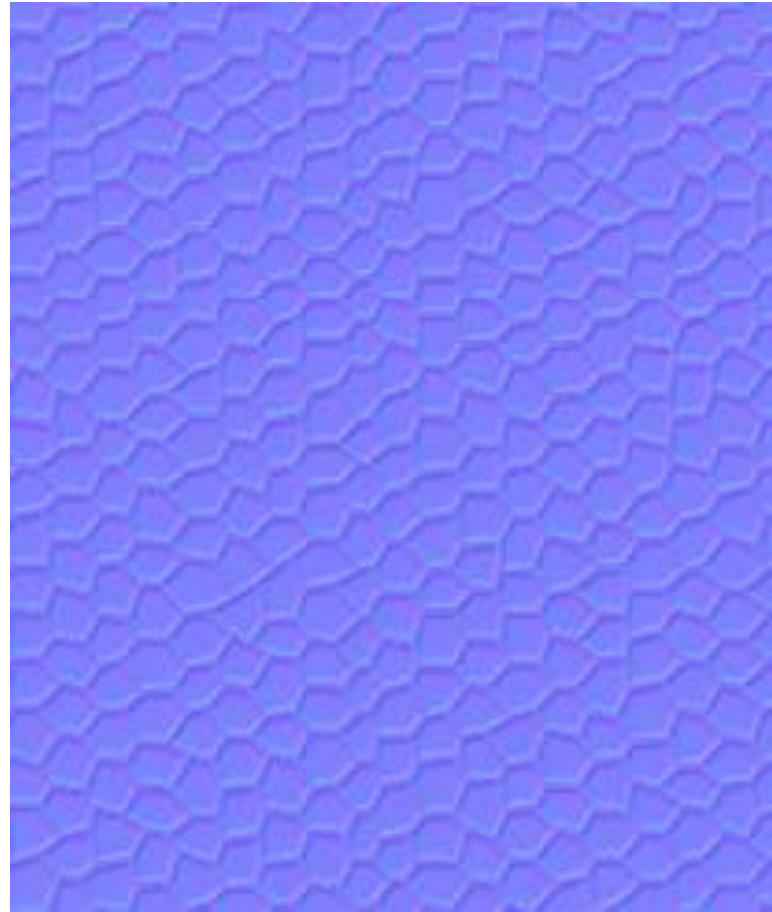
Si l'angle est supérieur à 90° cela signifie que la source est derrière la face. La face n'est donc pas éclairée.

Composantes nécessaires pour calculer la lumière



Normal map

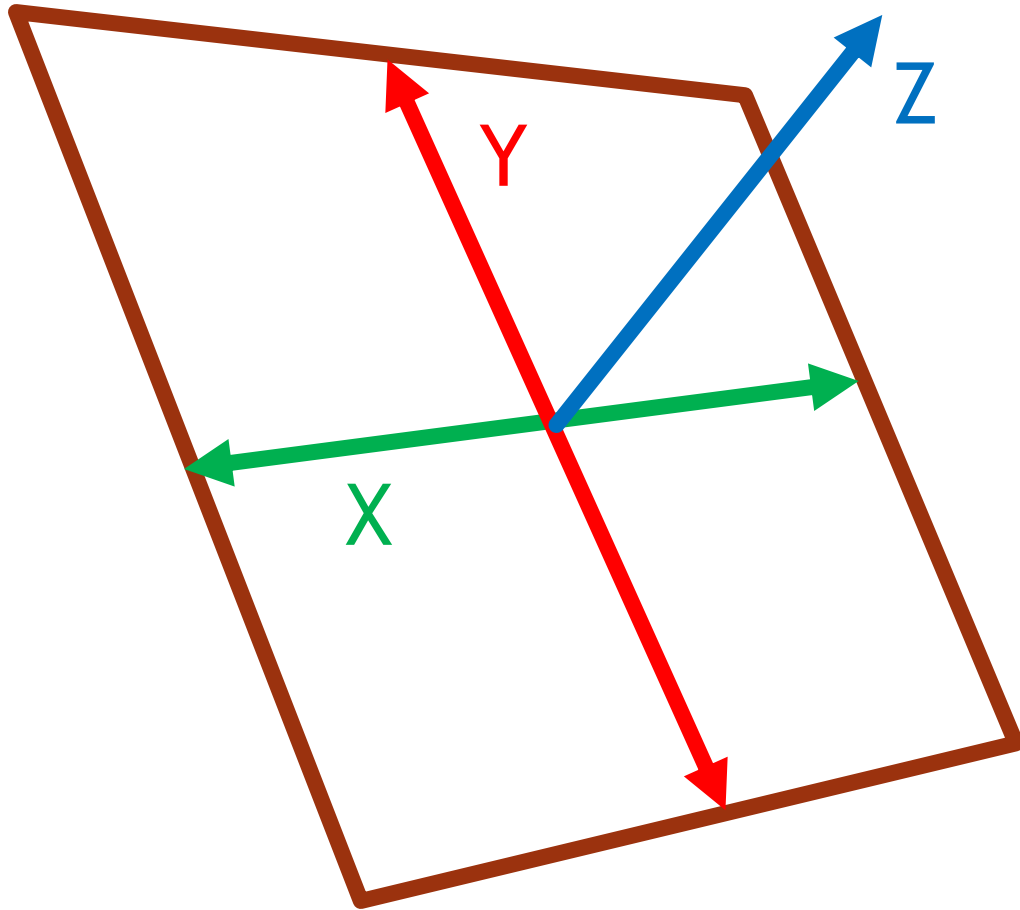
C'est ici que prend tout son sens une « normal map ». Cela permet de changer l'illumination d'une surface sans ajouter de polygone.



Code couleur normal map

- ▶ Pour x et y, valeur possible entre 0 et 255
 - ▶ 0 := -1
 - ▶ 128 = 0
 - ▶ 255 = +1
- ▶ Rouge donne la composante X
- ▶ Vert donne la composante Y
- ▶ Bleu donne la composante Z
 - ▶ Z entre 0 et -1(de 128=0 à 255=-1) donc la valeur minimale du bleu est toujours 128 donc c'est pourquoi la texture semble bleue

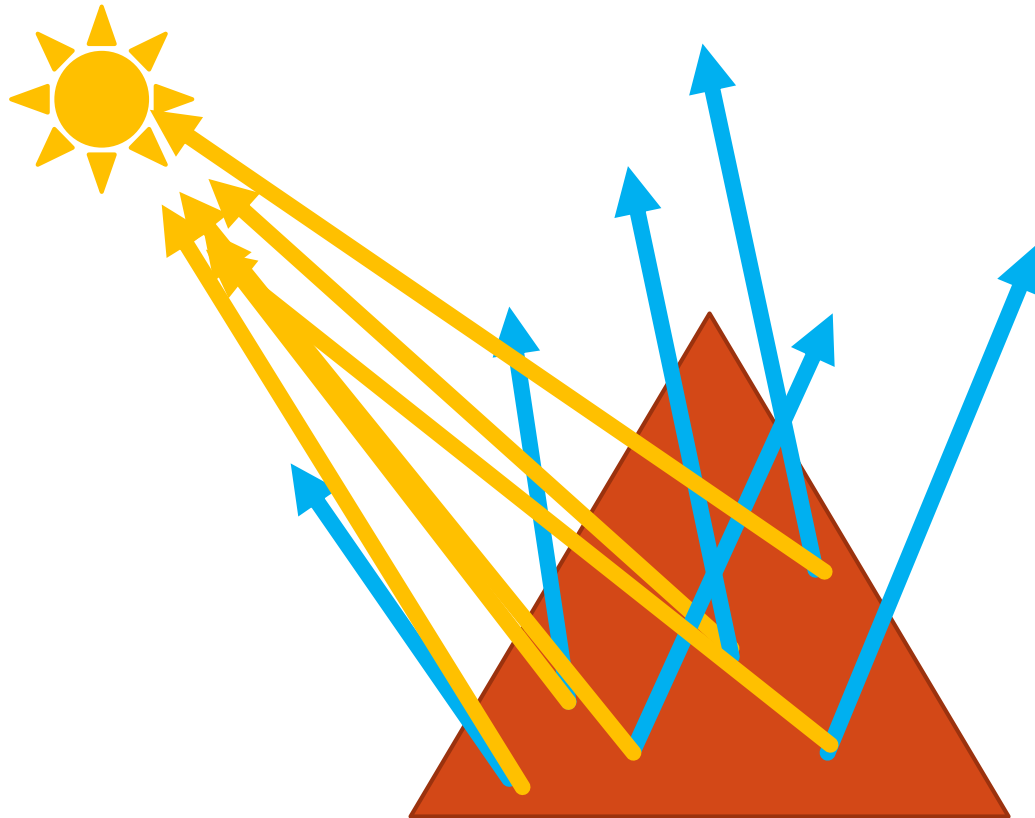
Normal map



Normal map

Chaque pixel d'une normale map définit une nouvelle normal pour chaque pixel d'un polygone. Lors du calcul de la couleur, chaque pixel du polygone donnera donc une valeur différente de luminosité.

Composantes nécessaires pour calculer la lumière



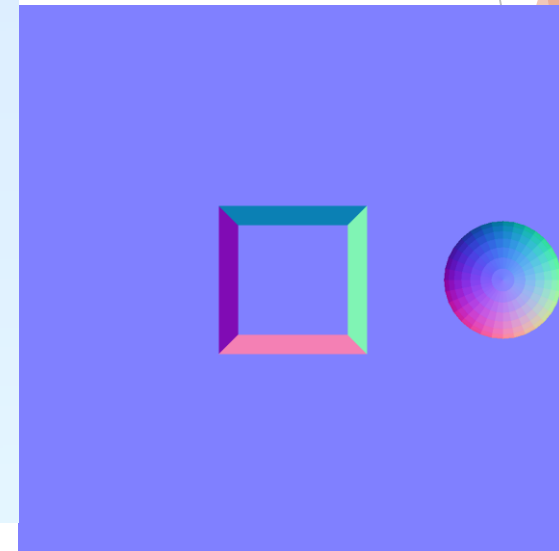
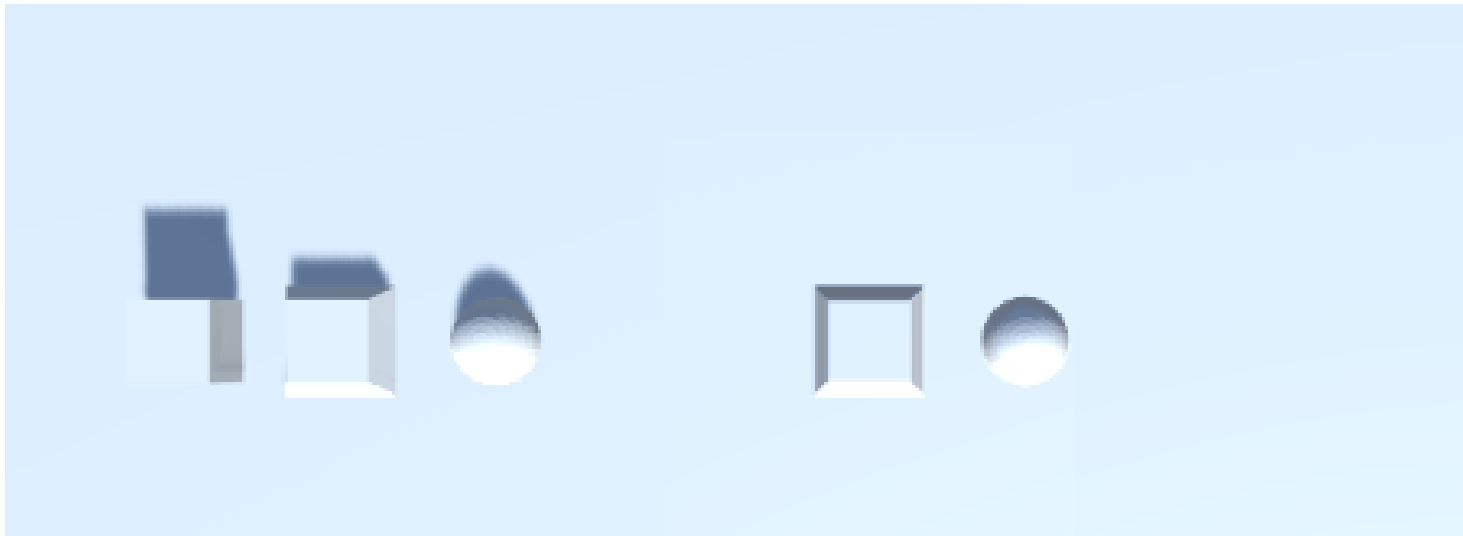
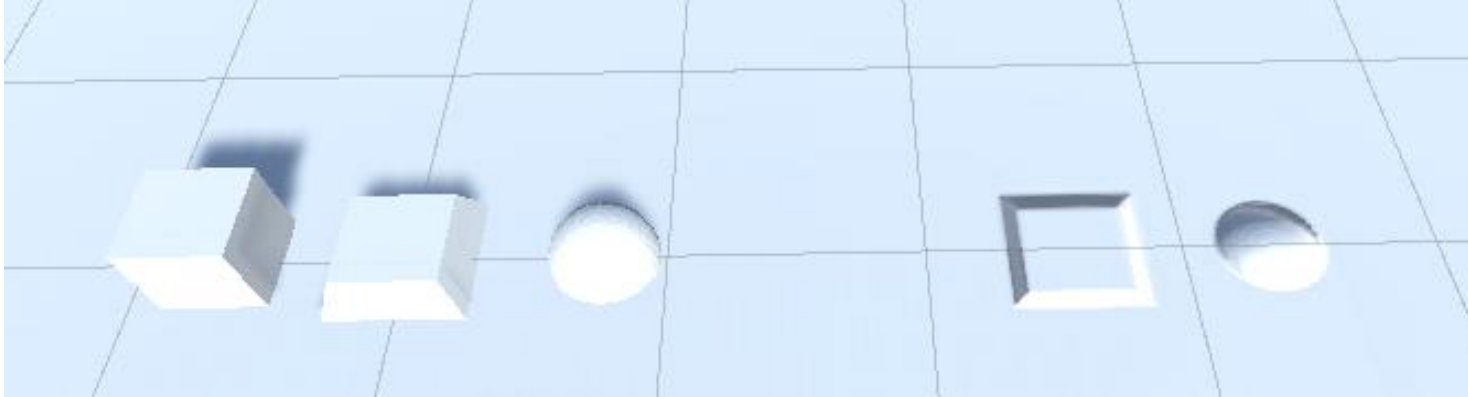
Avantage des normales

Coût extrêmement faible pour les performances, mais ajoute des détails aux meshes.

Faiblesse des normales map

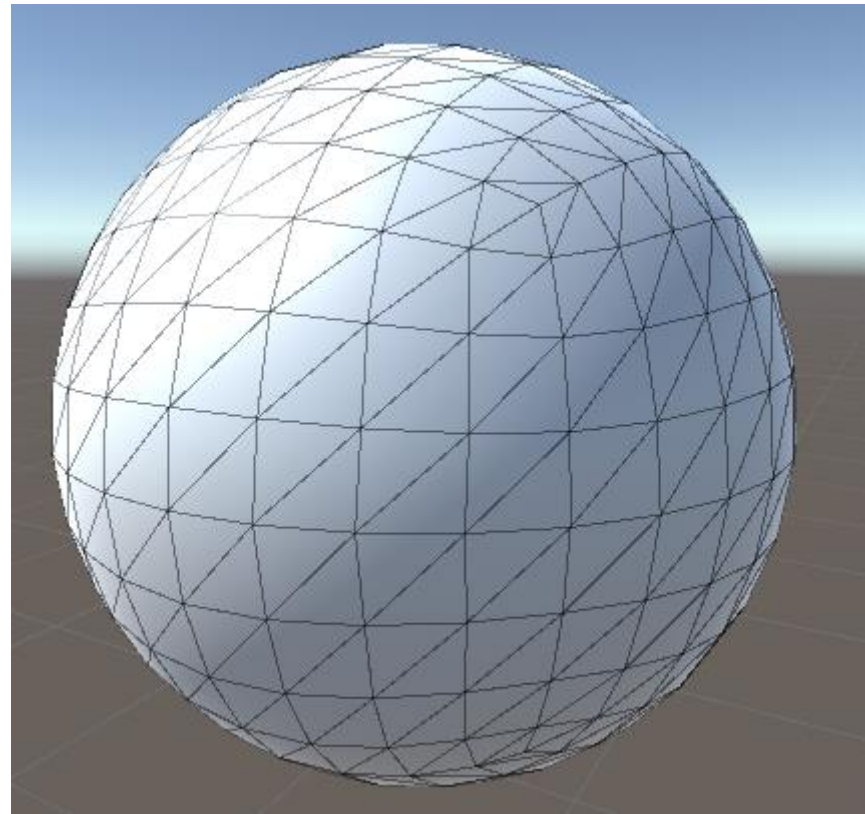
Lorsqu'inspecté de près l'illusion sera brisé.

Example

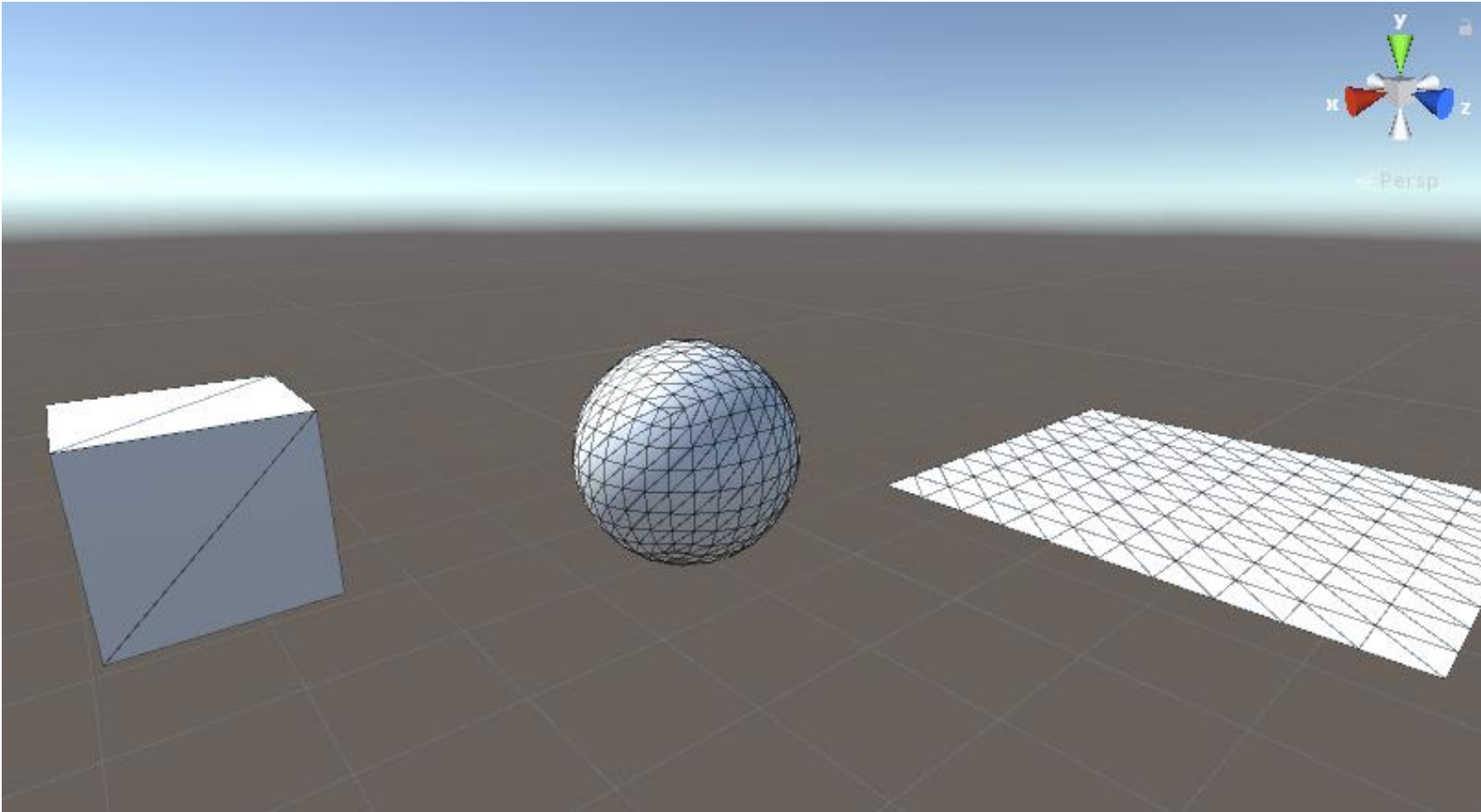


Pour définir une forme en 3D

Généralement, le polygone utilisé sera le triangle.



Les sommets, vertex ou vertices

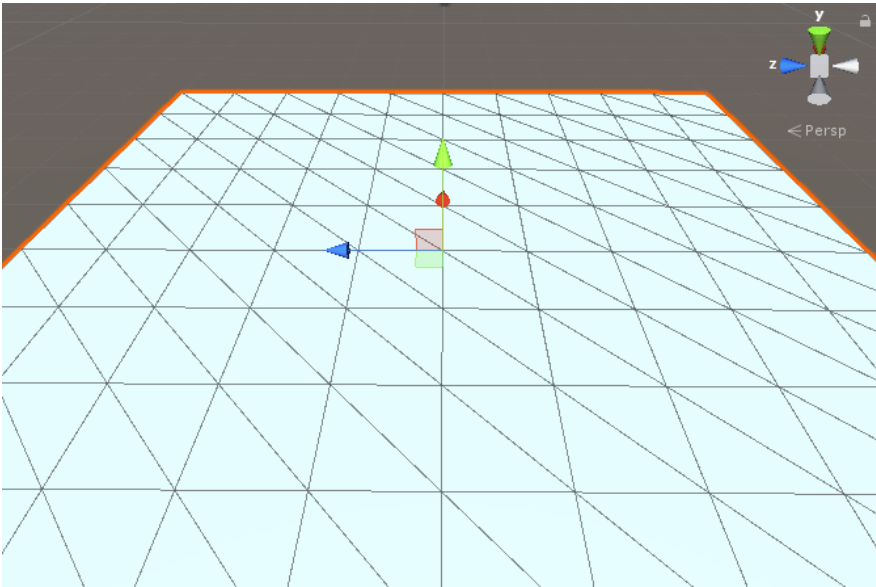


Tessellation

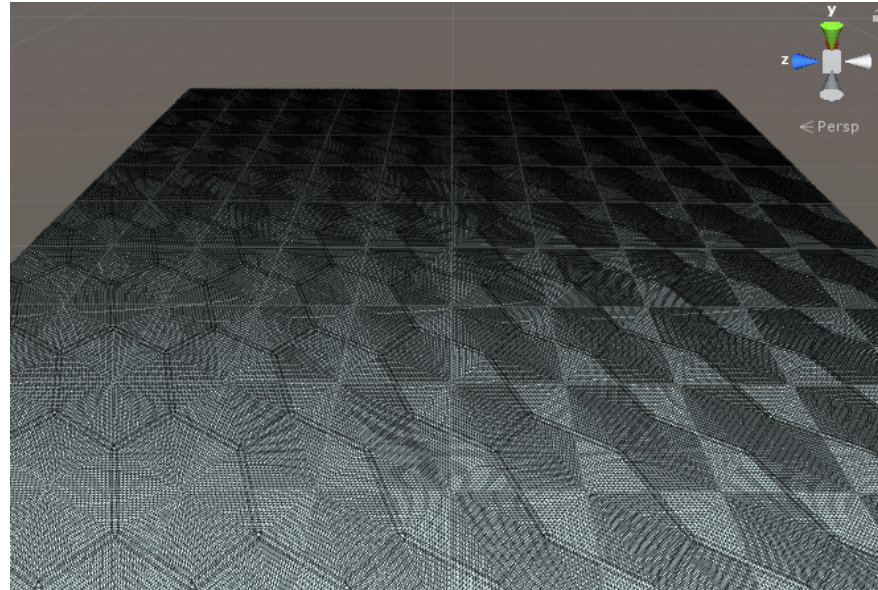
C'est le pavage d'un objet sans chevauchement et sans trou. En jeux vidéo, le nombre de polygones présent dans la tessellation peut être contrôlé. On augmente généralement le nombre de polygones présent dans un objet près de la caméra.

Exemple de tessellation

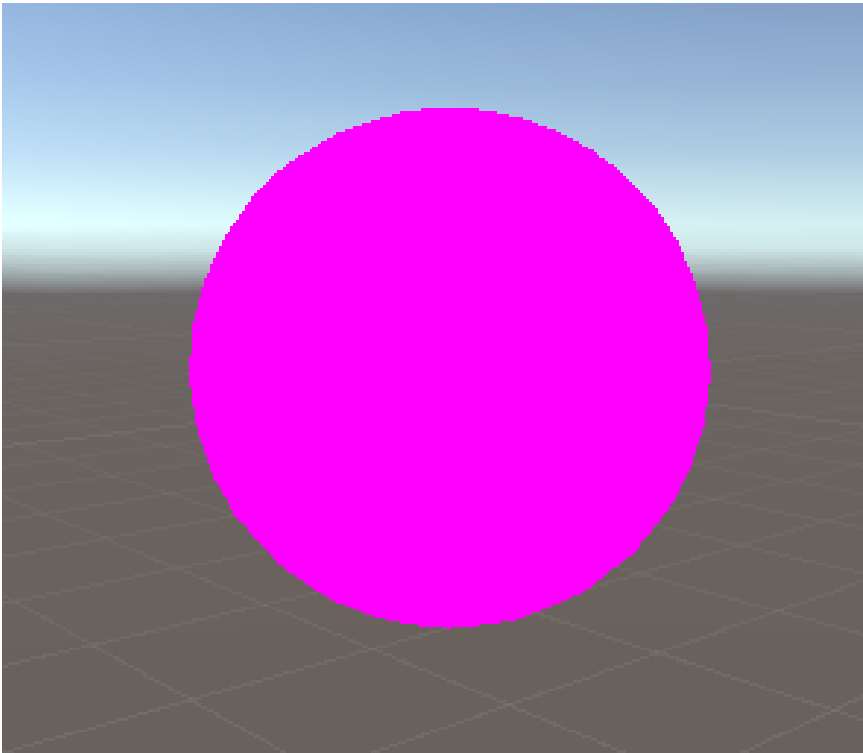
Tesselation avec peu de polygones



Tesselation avec beaucoup de polygones

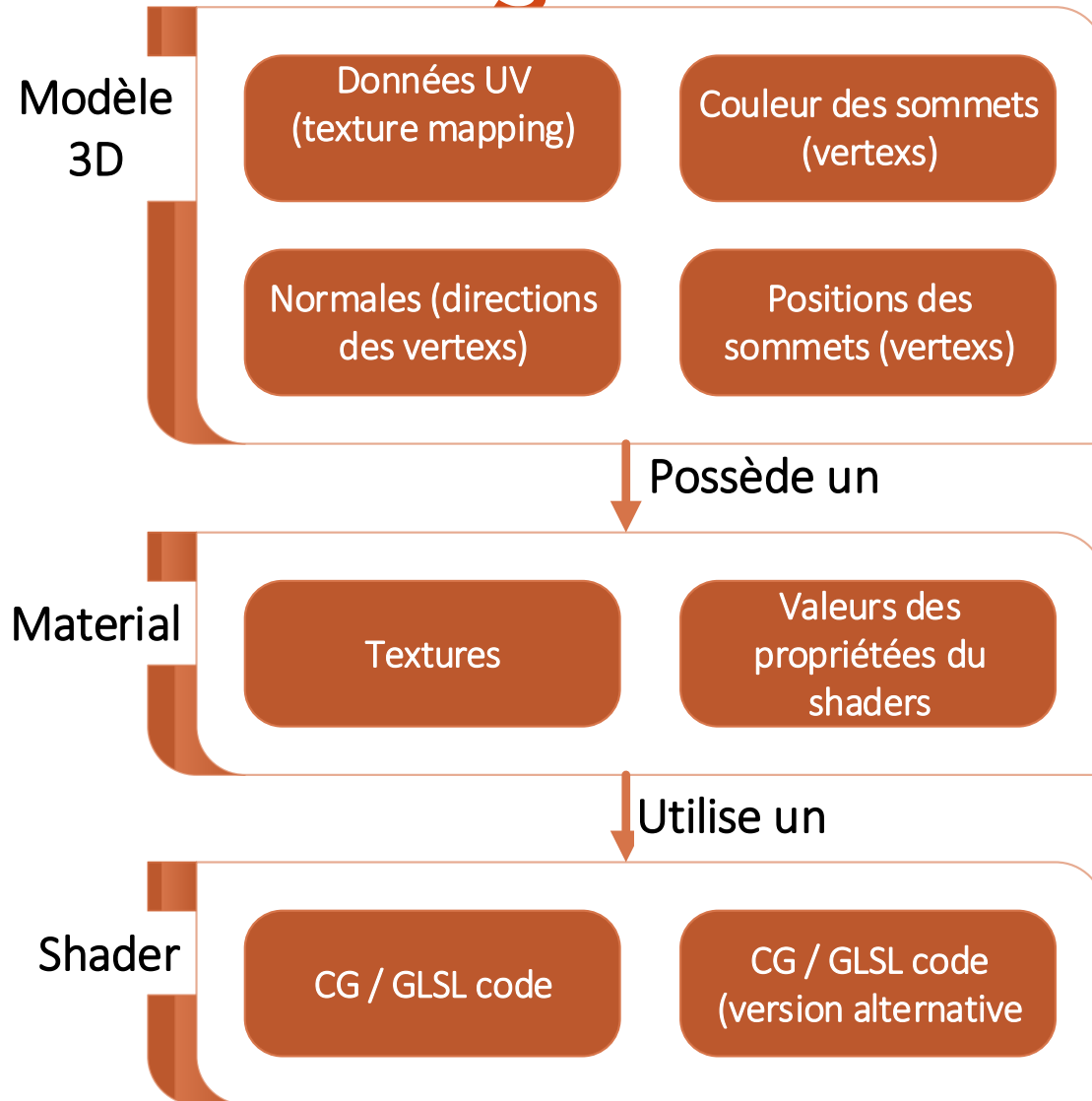


Sphère (3D) avec problème de shader



Par défaut, si un shader est absent ou ne compile pas, l'objet sera affiché avec une couleur rose. La couleur rose est en faite un shader qui retourne un pixel rose partout.

Le « Rendering Process »



Où s'exécute le shader ?

C'est la carte graphique qui exécute les calculs. Un shader est un programme fait spécifiquement pour fonctionner sur un GPU.

Les types de shaders dans Unity

- ▶ Surface Shader
- ▶ Unlit Shader
- ▶ Post processing Shader

Mais on peut passer d'un à l'autre facilement.

Surface shader

C'est un shader dans Unity qui permet une interaction avec la lumière. C'est le shader le plus commun.

Unlit Shaders

C'est un shader qui n'interagit pas avec la lumière.

(post processing)Image Effect Shaders

C'est un shader pour un post traitement sur une image. On peut donc : prendre l'image de la caméra et la transformer en nuance de bleu. Ce shader ne tient pas compte des vertex et des polygones.

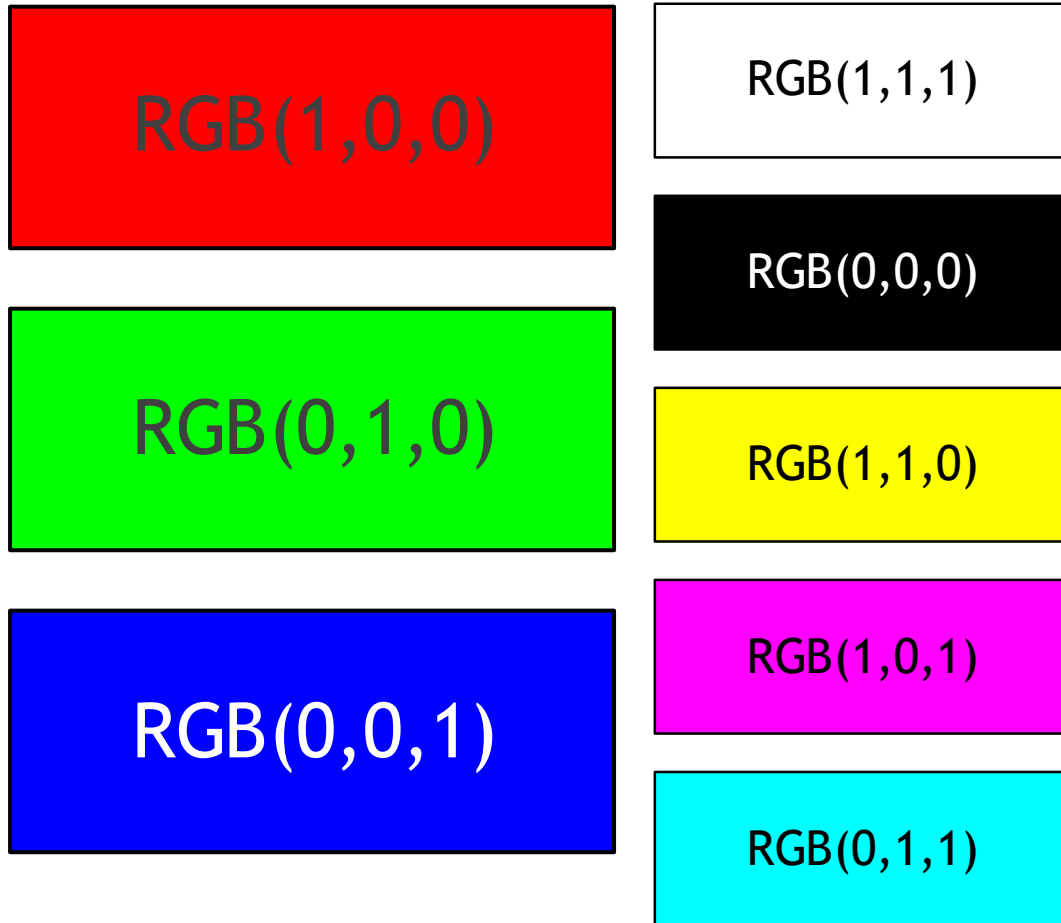
Les mathématiques des couleurs

- ▶ Le format RGB
- ▶ L'addition de couleur
- ▶ La soustraction de couleur
- ▶ La multiplication de couleur
- ▶ La division de couleur
- ▶ Les puissances

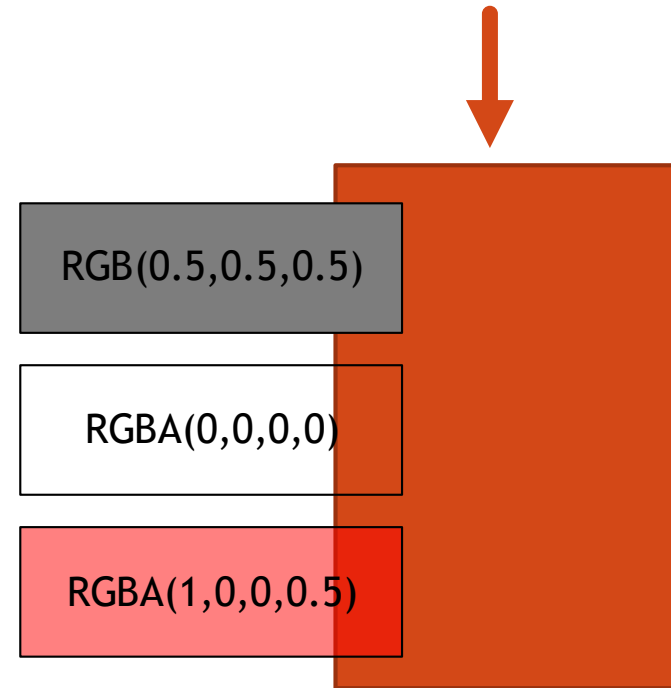
Le format RGB et RGBA

- ▶ Chaque composante du RGB et RGBA possède une valeur entre 0 et 1.
- ▶ **R** **G** et **B** représentent le Rouge (**R**ed) le vert (**G**reen) et le Bleu (**B**lue) respectivement
- ▶ Le A dans RGBA représente le canal Alpha et donc la transparence.

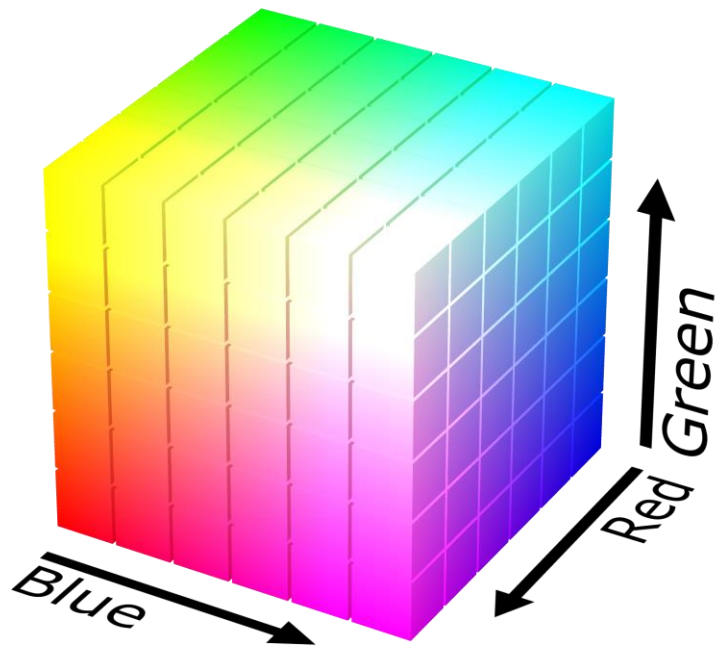
Exemple de RGB



Fond pour illustration de la transparence

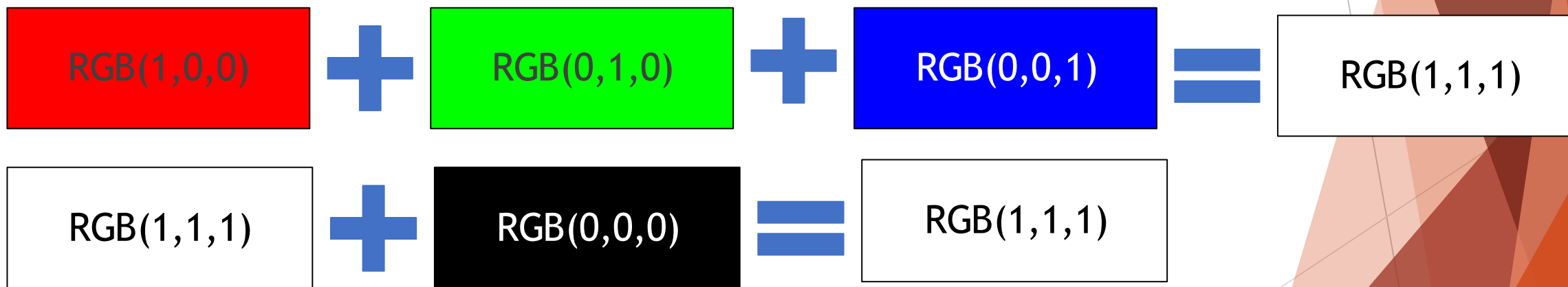


Le cube RGB

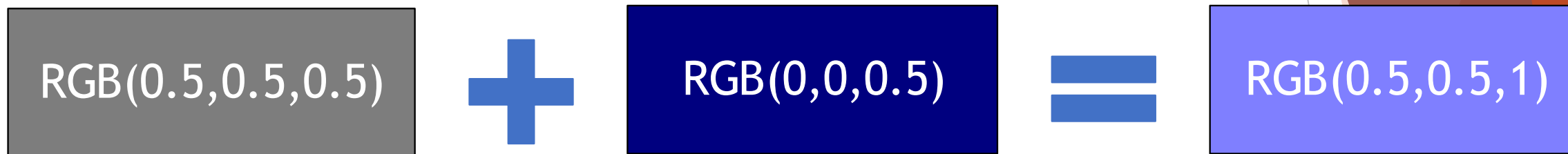


L'addition de RGB

L'addition va toujours résulter en une couleur plus claire.


$$\begin{array}{l} \text{RGB}(1,0,0) + \text{RGB}(0,1,0) + \text{RGB}(0,0,1) = \text{RGB}(1,1,1) \\ \text{RGB}(1,1,1) + \text{RGB}(0,0,0) = \text{RGB}(1,1,1) \end{array}$$

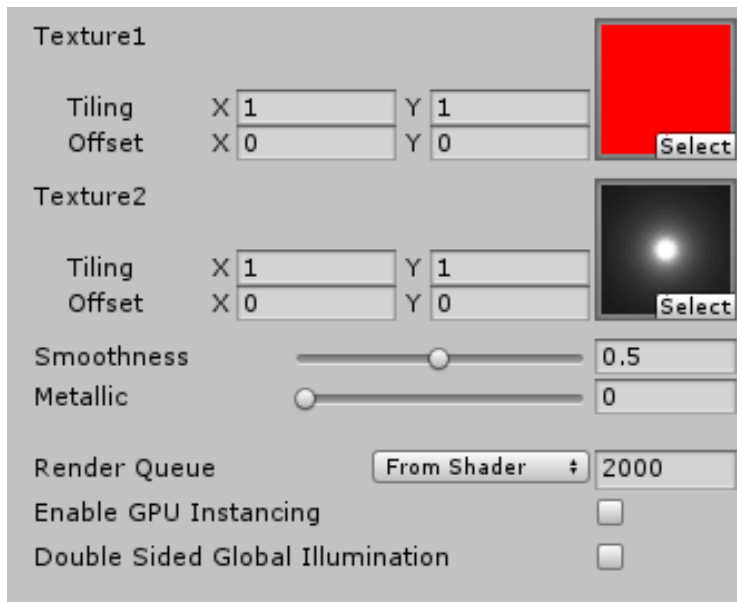
Addition (2)



The diagram illustrates the addition of two RGB colors. On the left, a gray box contains the text "RGB(0.5,0.5,0.5)". To its right is a large blue plus sign. Further right is a dark blue box containing the text "RGB(0,0,0.5)". To the right of this box is a blue equals sign. Finally, on the far right, a light blue box contains the text "RGB(0.5,0.5,1)".

$$\text{RGB}(0.5, 0.5, 0.5) + \text{RGB}(0, 0, 0.5) = \text{RGB}(0.5, 0.5, 1)$$

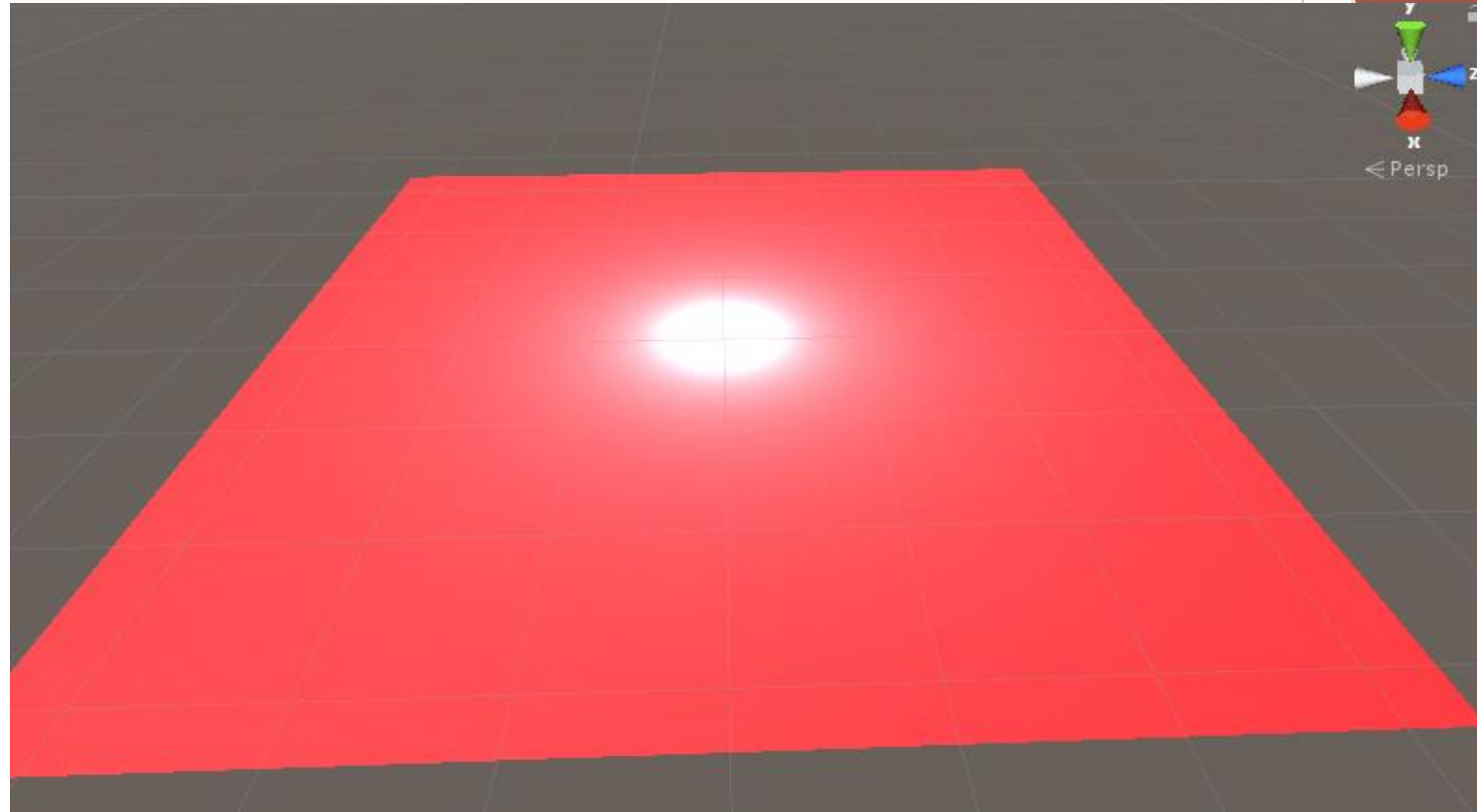
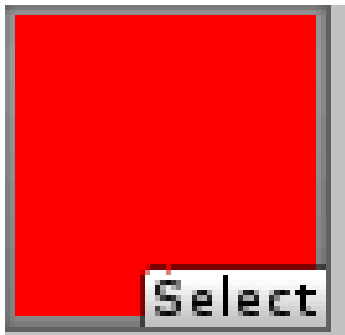
L'addition de texture



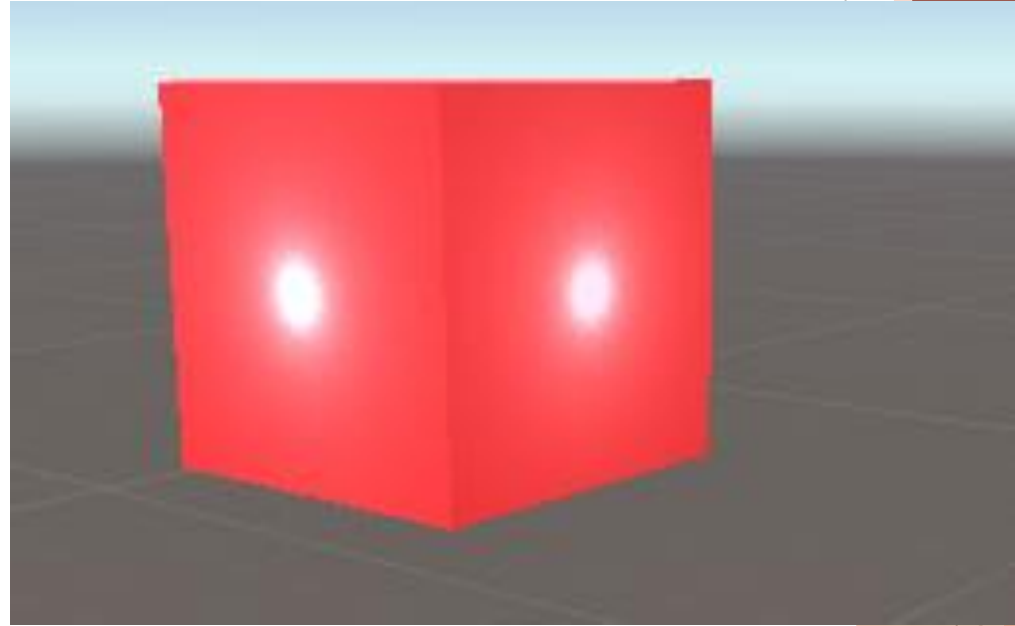
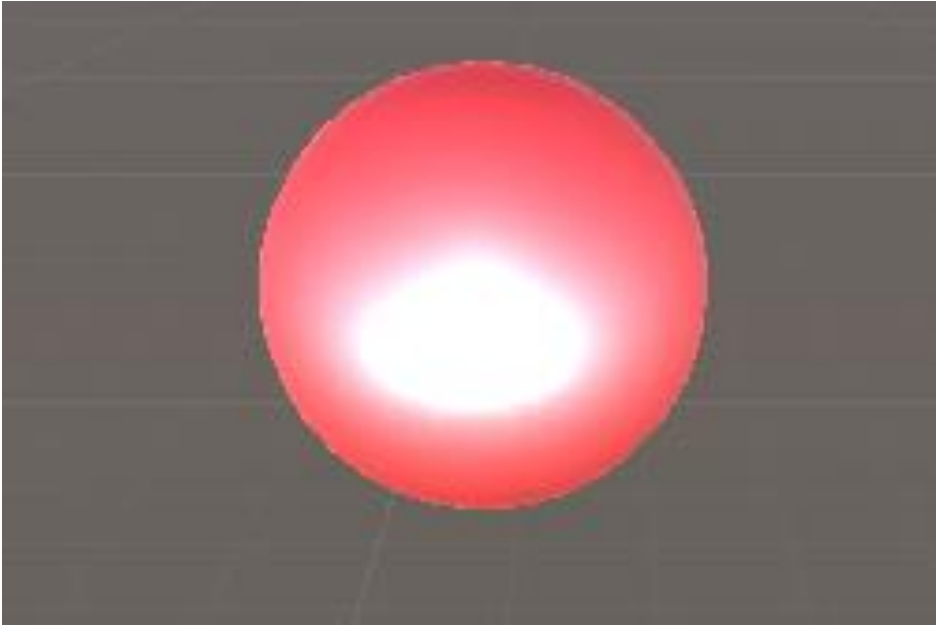
```
fixed4 c = tex2D (_Tex1, IN.uv_Tex1) + tex2D(_Tex2, IN.uv_Tex1);  
o.Albedo = c.rgb;  
  
o.Metallic = _Metallic;  
o.Smoothness = _Glossiness;  
o.Alpha = c.a;
```

Code pour addition de texture.

L'addition de texture



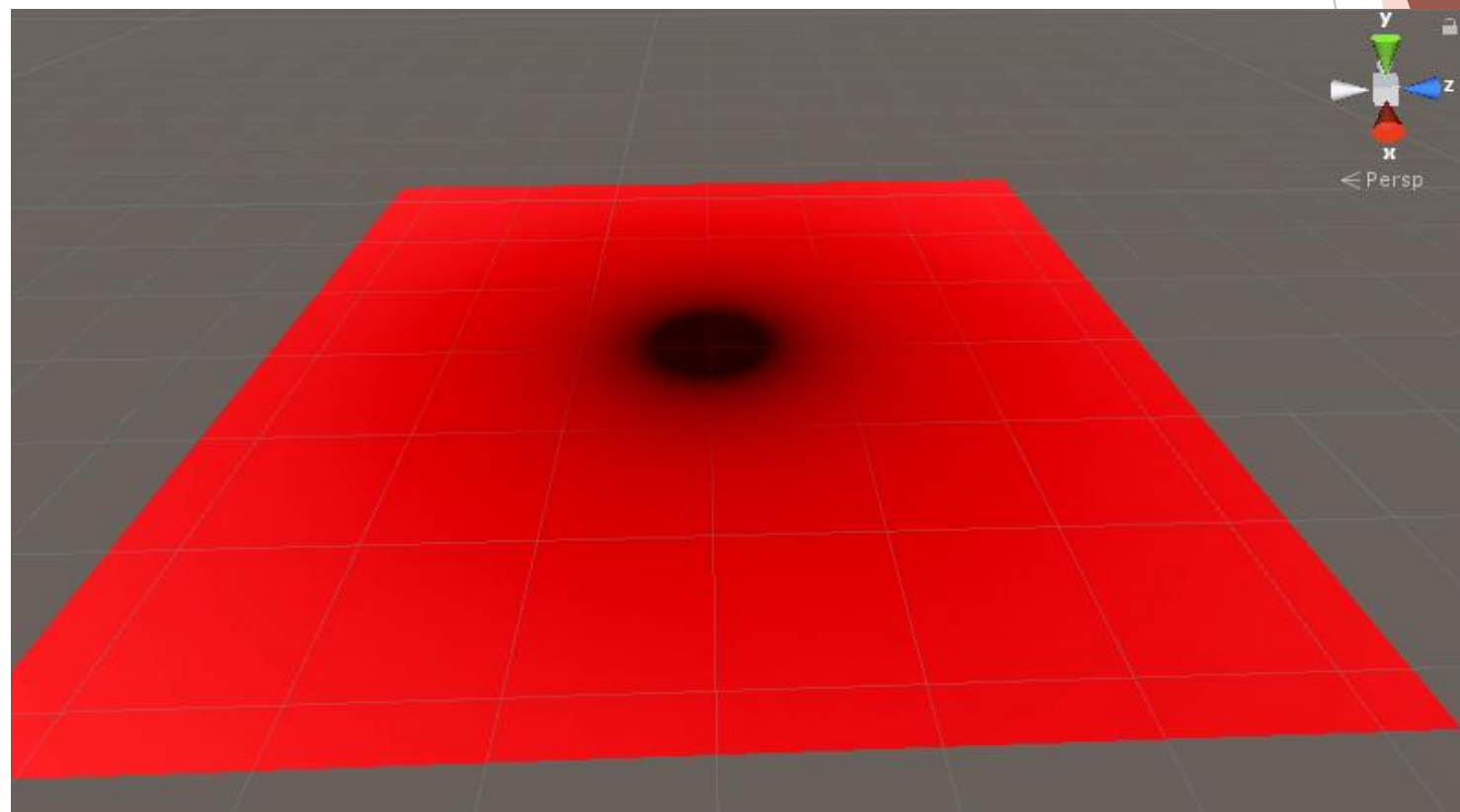
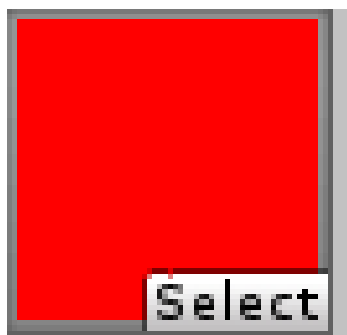
Avec une sphère



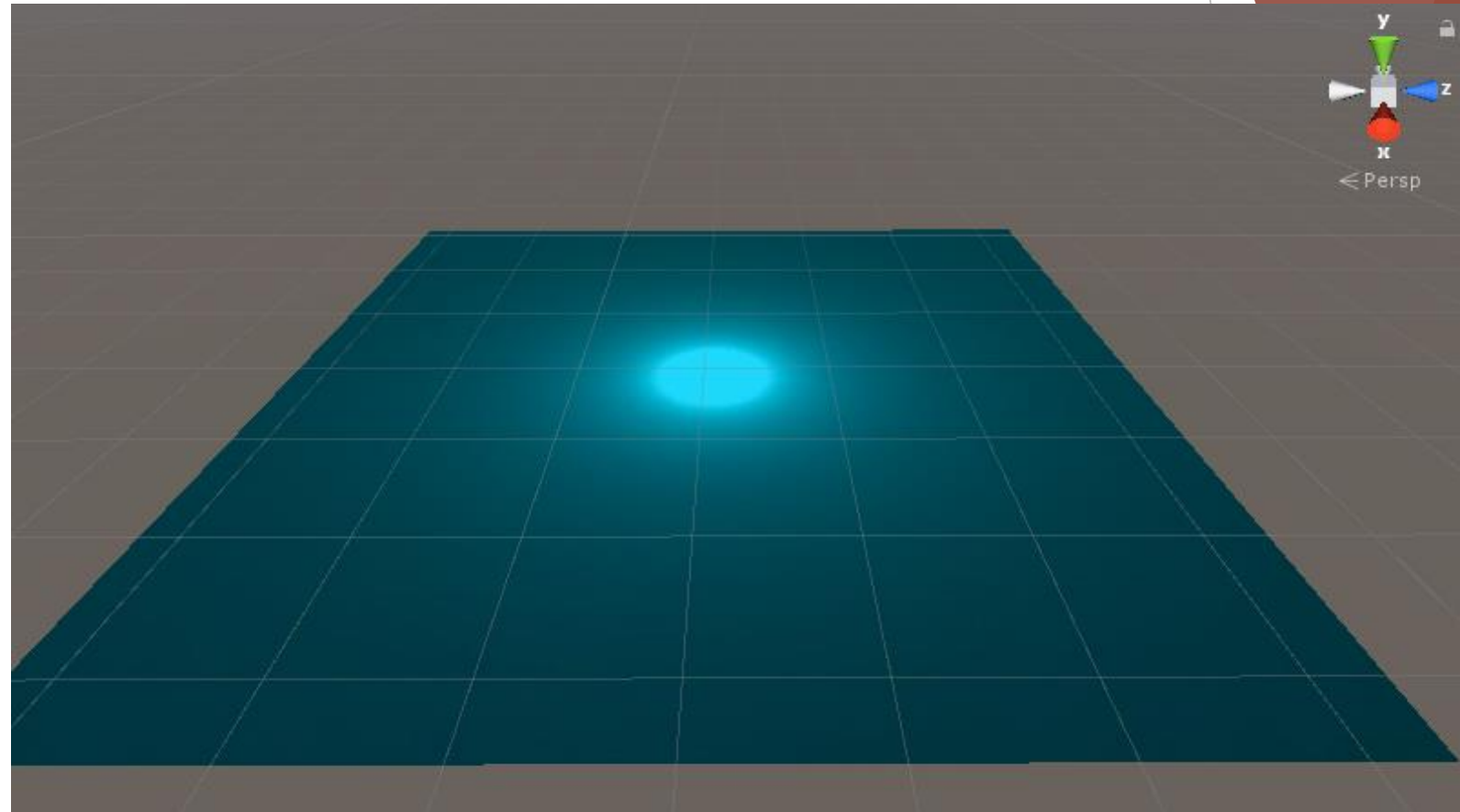
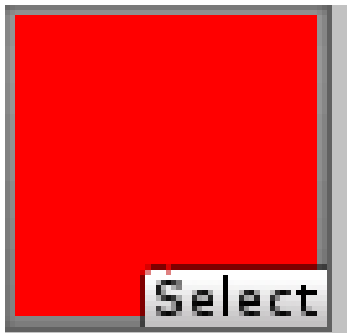
La soustraction

Fonctionne de la même manière que l'addition, mais les couleurs deviennent plus foncées. Dans l'addition comme la soustraction, les points de couleur noire $(0, 0, 0)$ n'ont aucun effet.

La soustraction de texture



La soustraction de texture



La multiplication

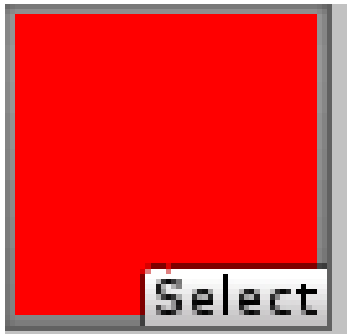
La multiplication des couleurs aura tendance à assombrir les couleurs. Il est toutefois possible de multiplier par une constante pour augmenter le contraste et la luminosité.

Exemple de multiplication

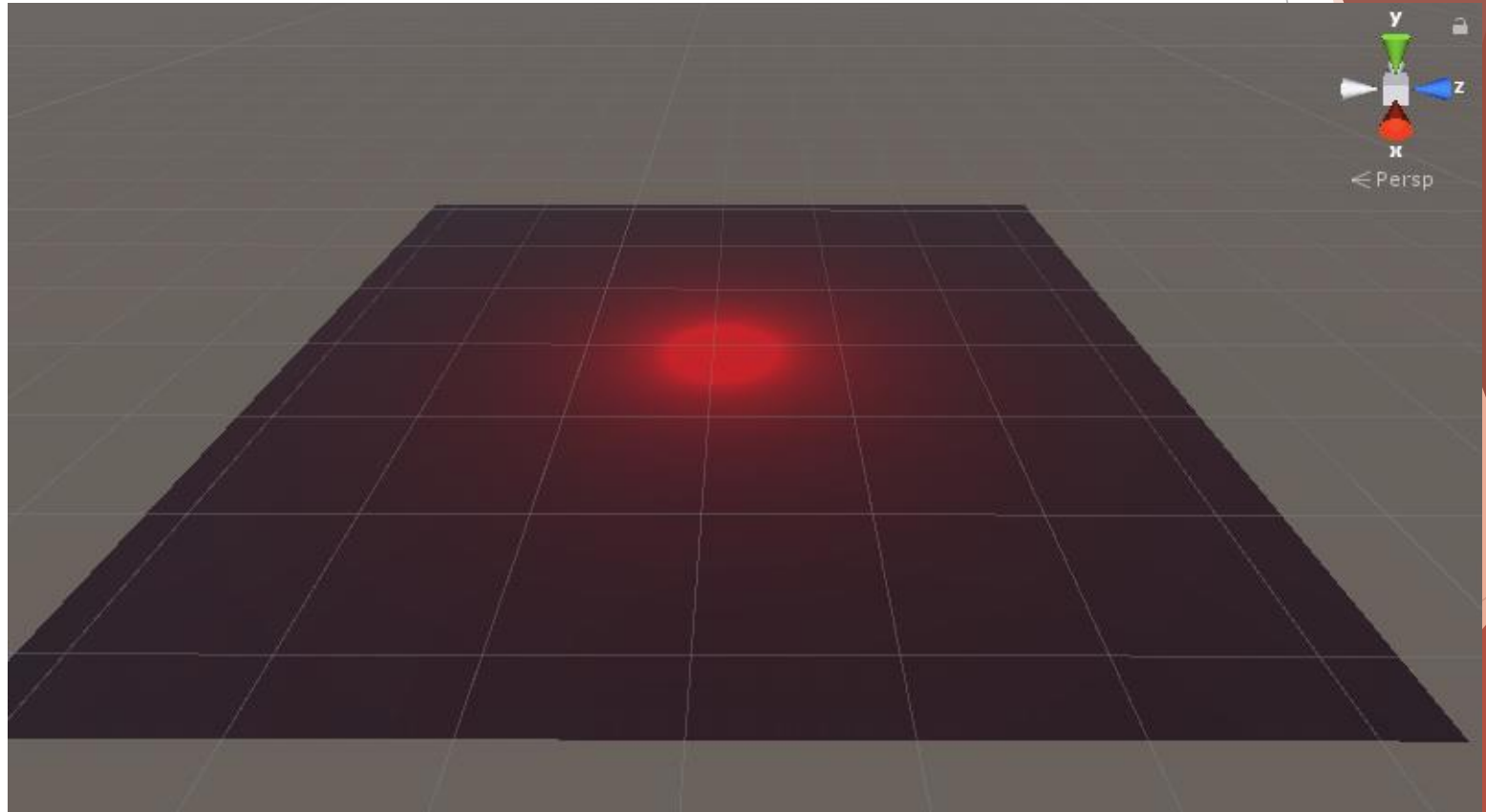
$$\text{RGB}(1,0,0) \times \text{RGB}(0.5,0.5,0.5) = \text{RGB}(0.5,0,0)$$

$$\text{RGB}(0.5,0,0) \times 2 = \text{RGB}(1,0,0)$$

La multiplication



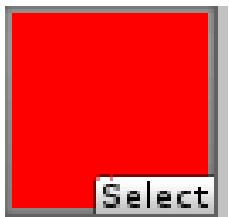
RGB (1, 0, 0)



La multiplication



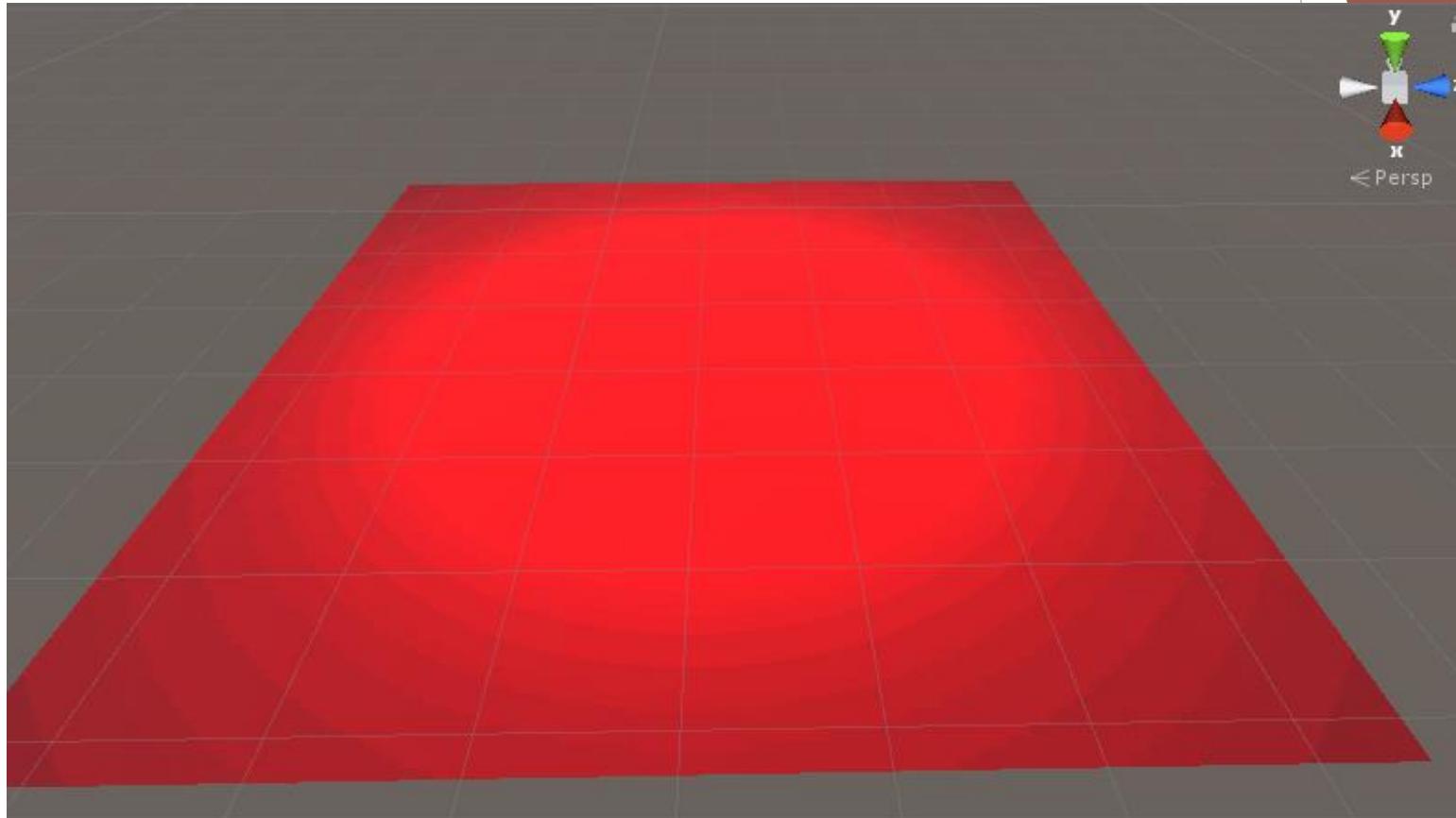
x



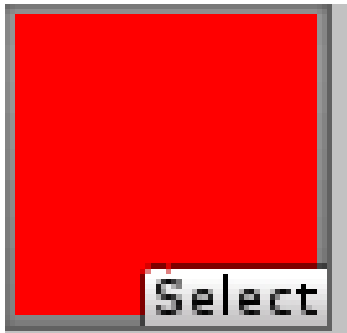
x

7

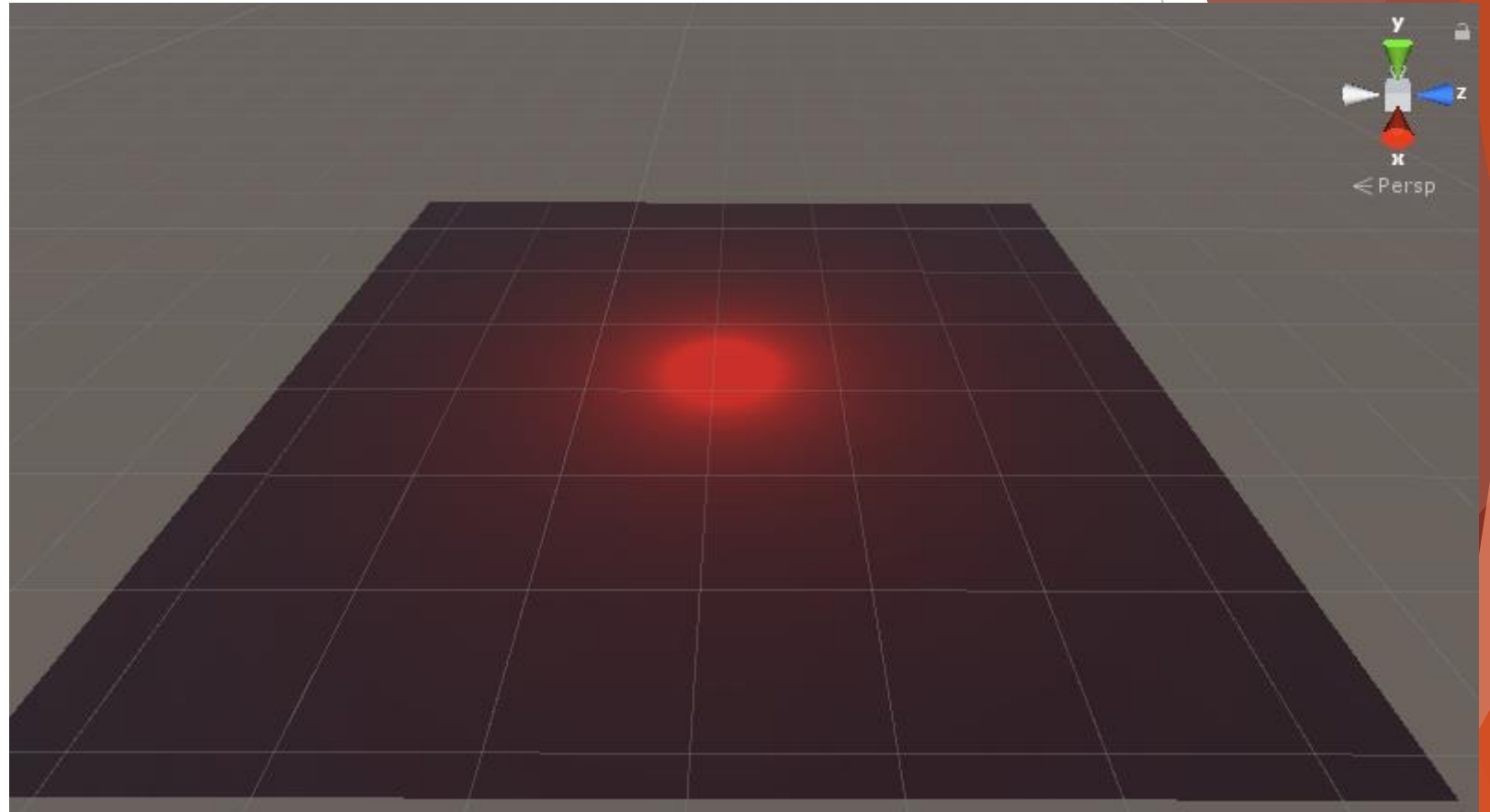
=



La multiplication avec une texture qui contient un peu de vert

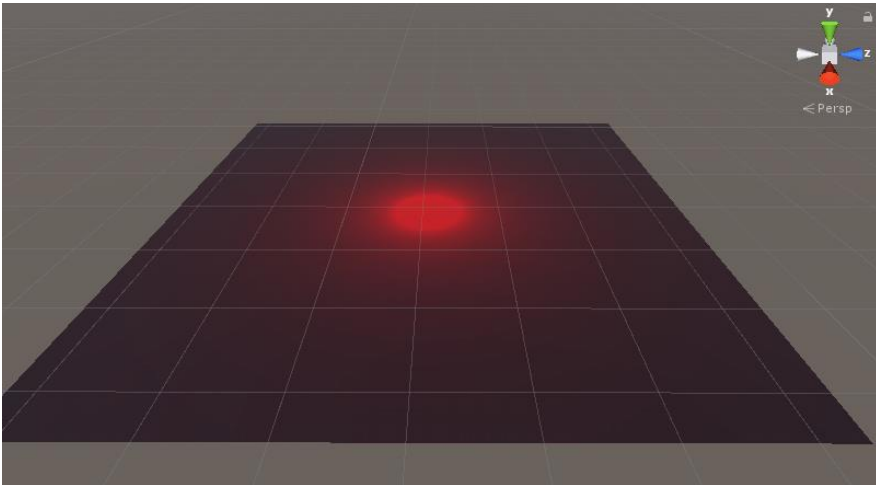


RGB(1,0.07,0)

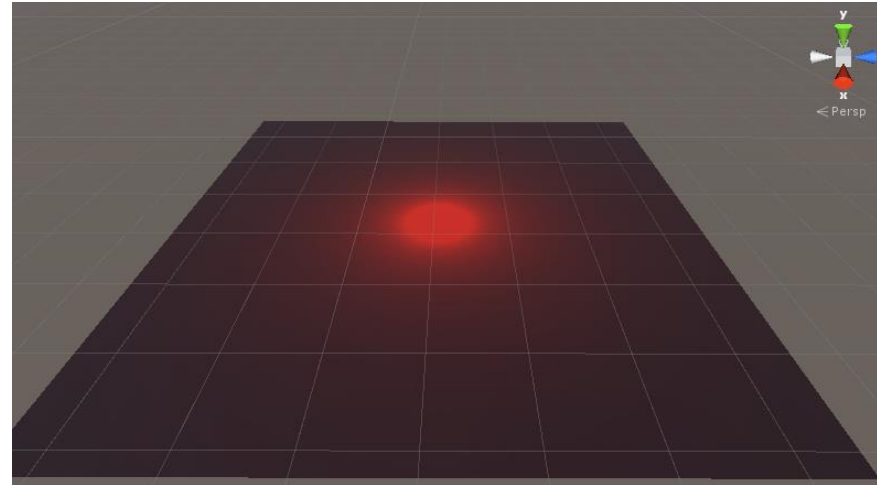


Aucune différence ou presque

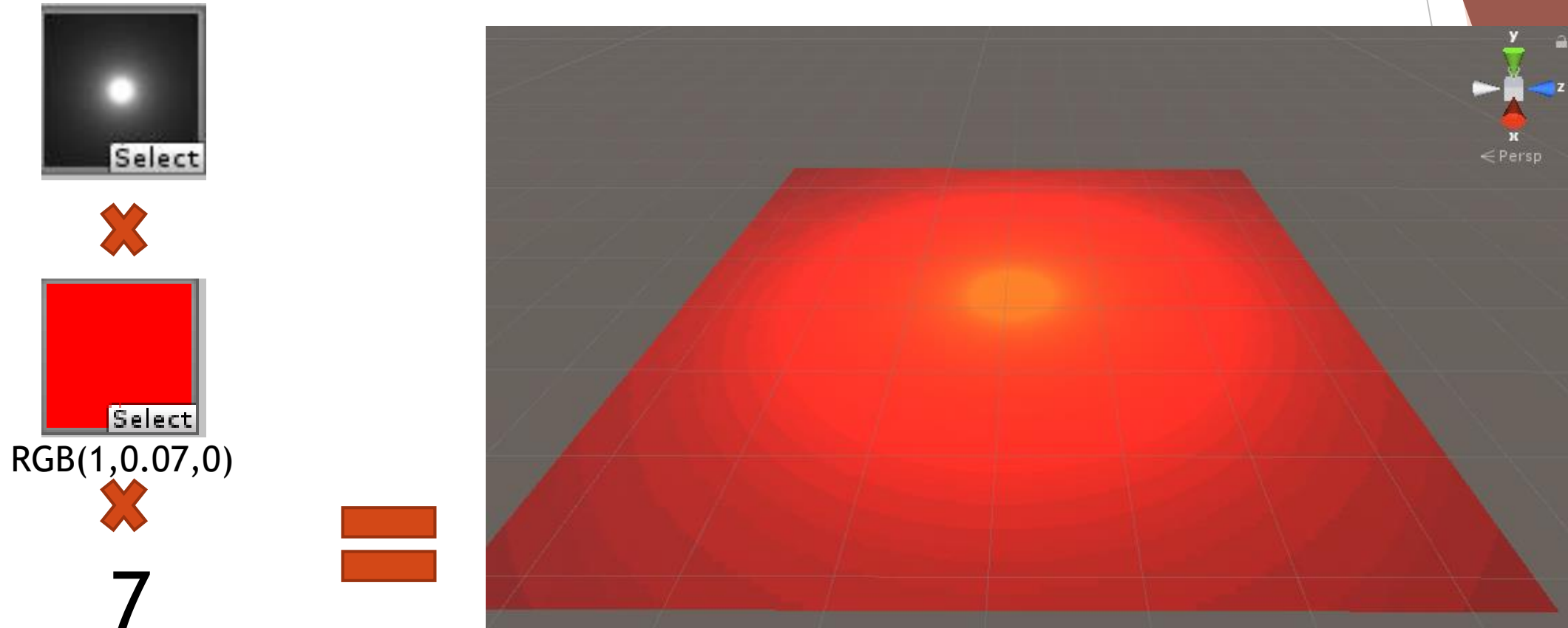
RGB (1, 0, 0)



RGB(1,0.07,0)



La multiplication avec une texture qui contient un peu de vert

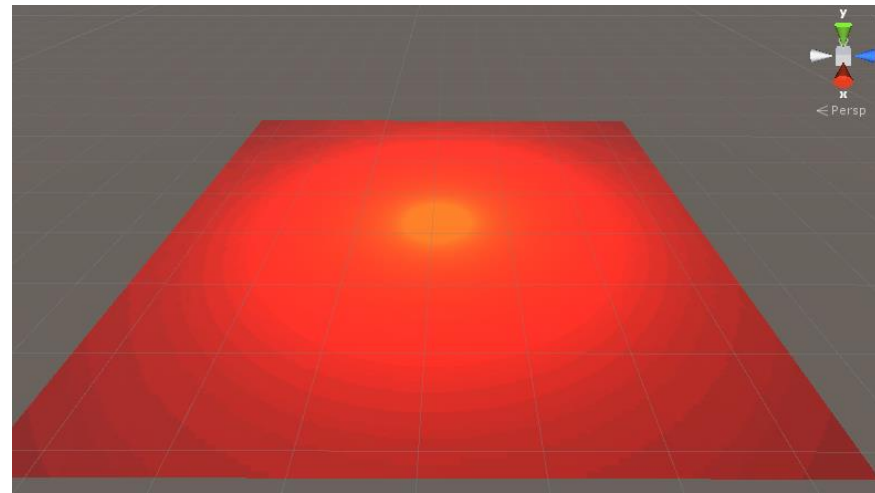
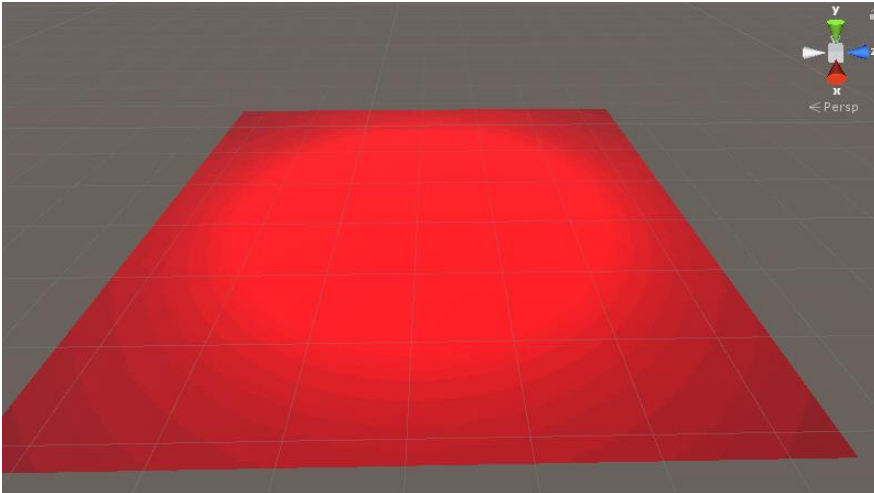


Avec constante de multiplication

Calcul avant multiplication de la texture 1

RGB (1, 0, 0) * 7 = RGB (7, 0, 0)

RGB(1,0.07,0) * 7 = RGB(7,0.49,0)



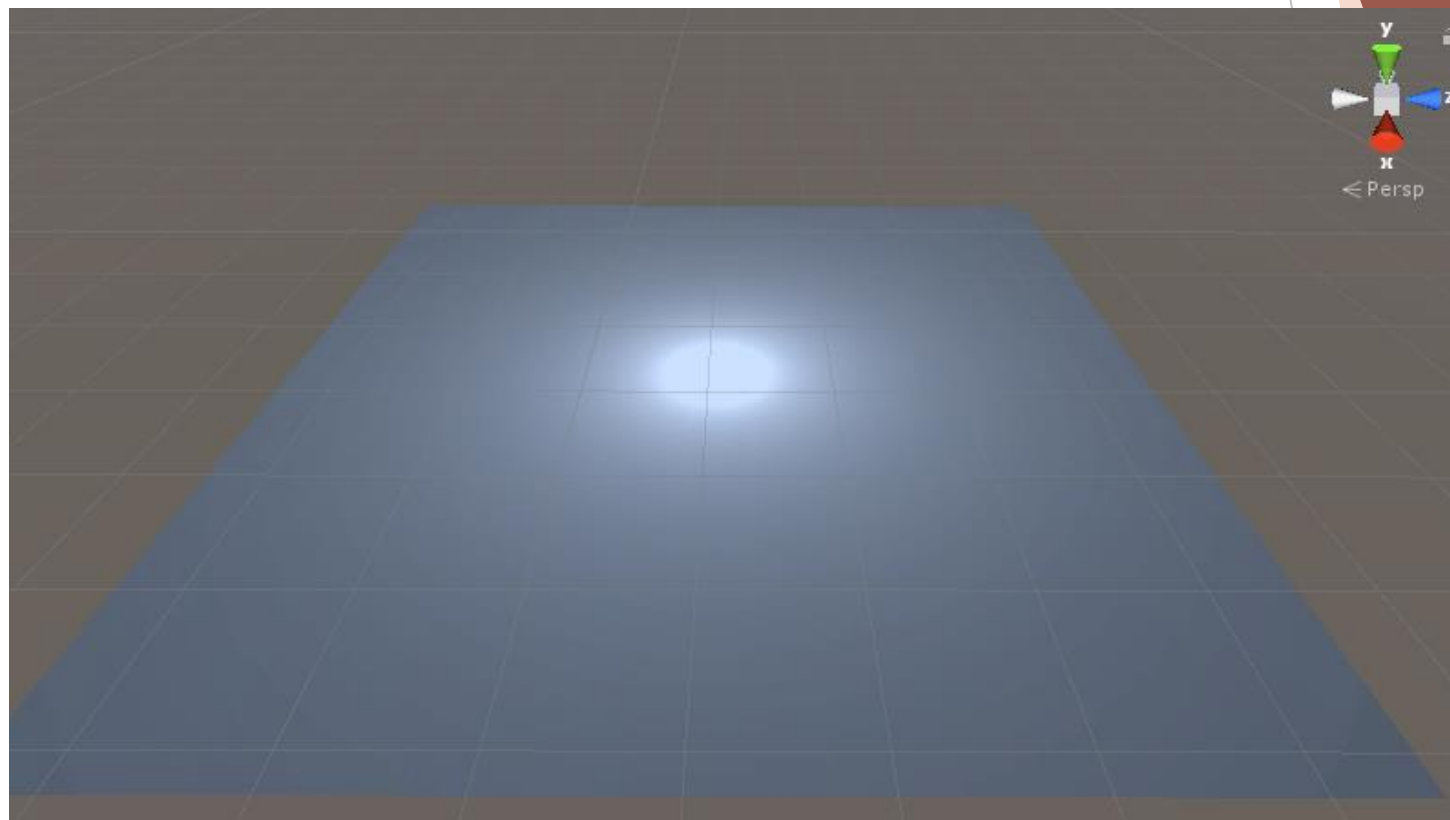
Division

Rarement pertinent : attention aux divisions par 0.

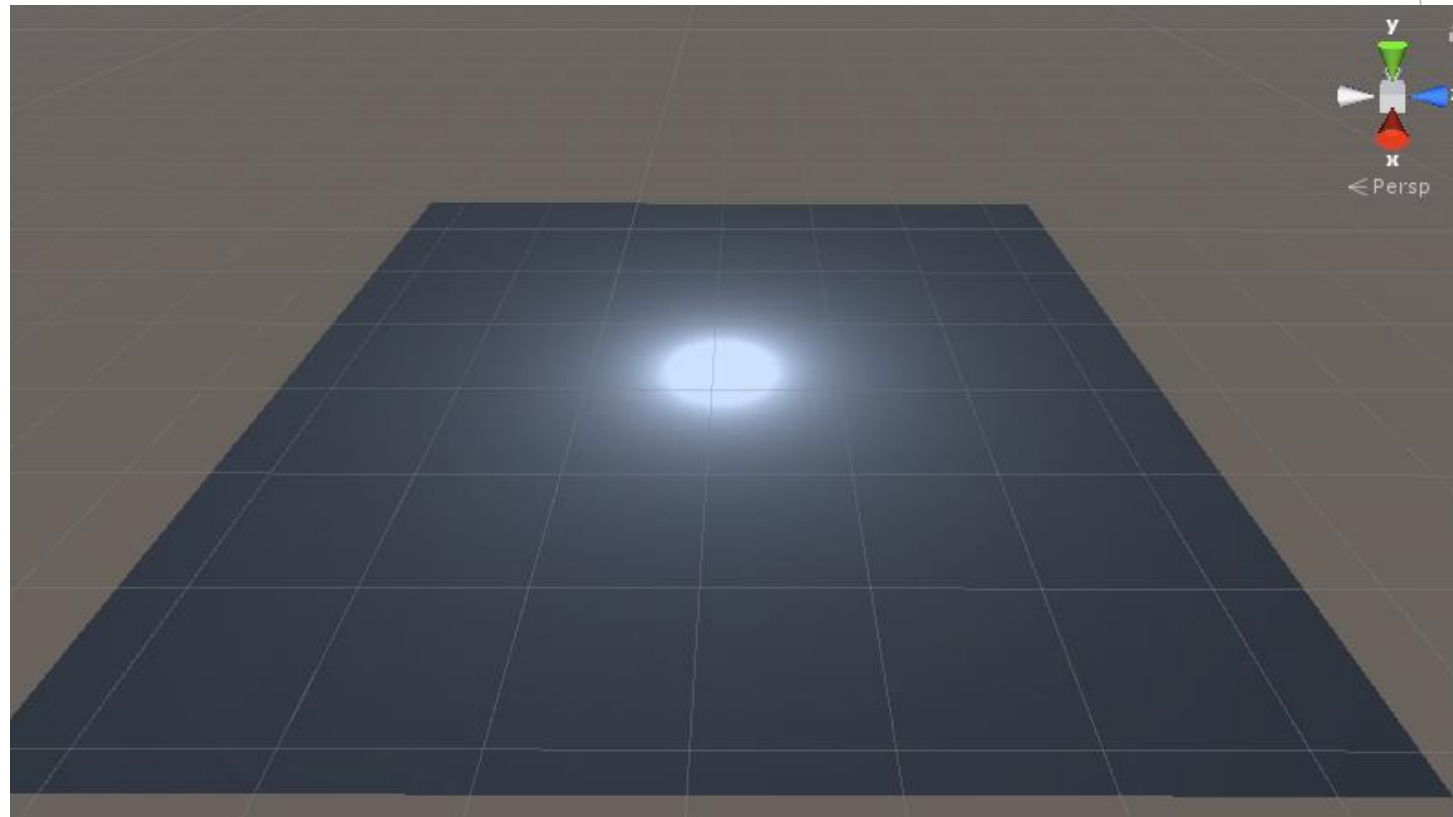
Puissance



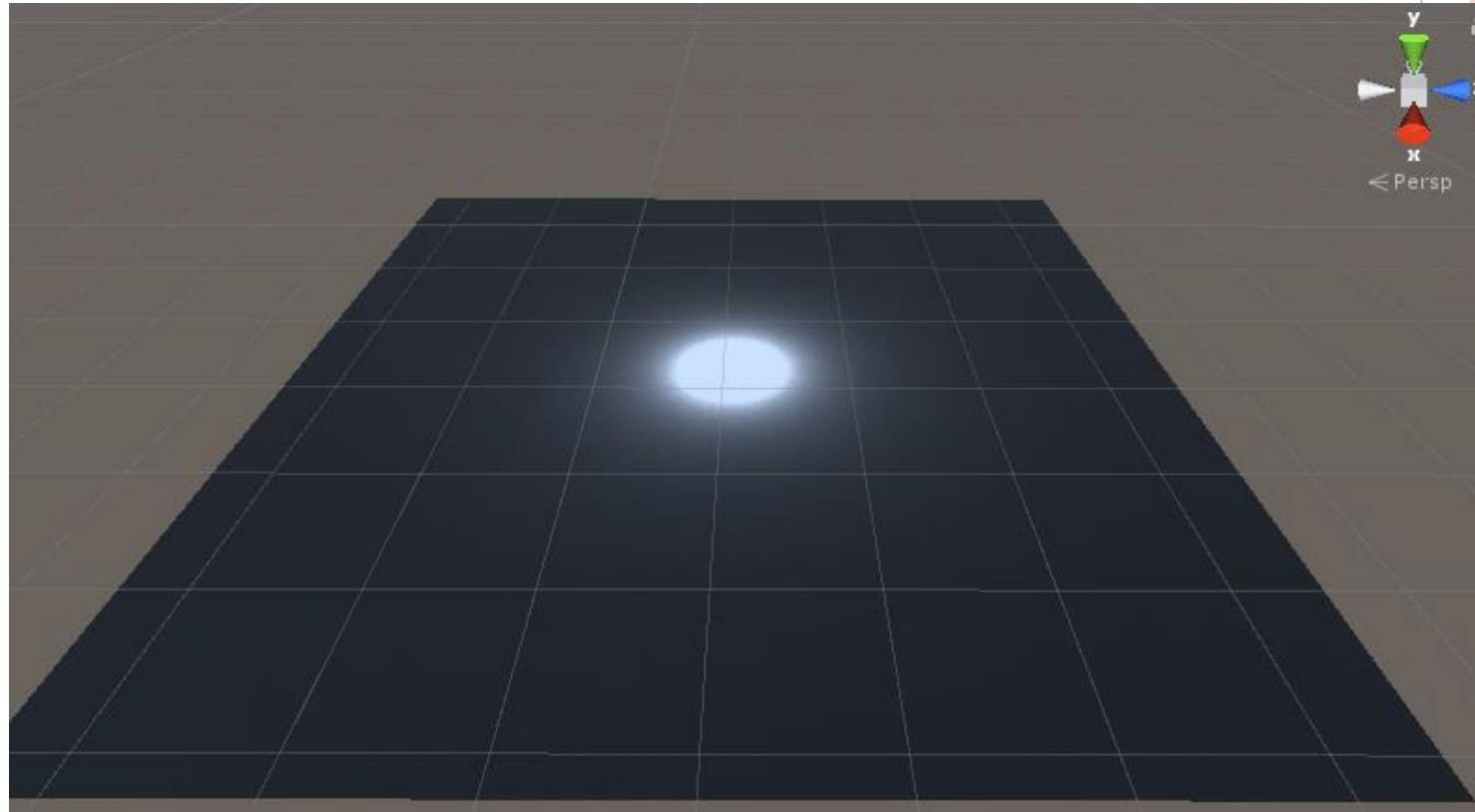
0.5



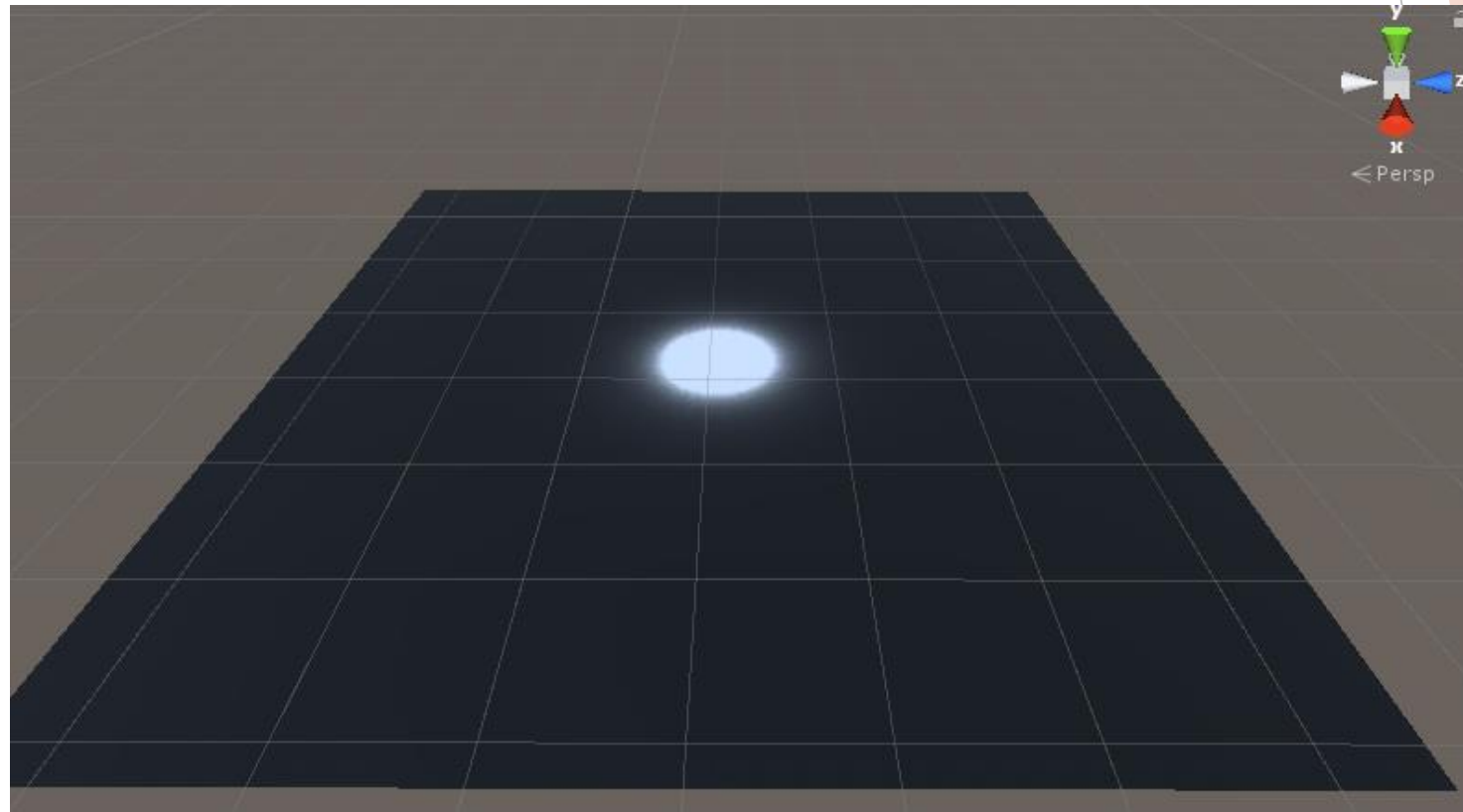
Puissance



Puissance

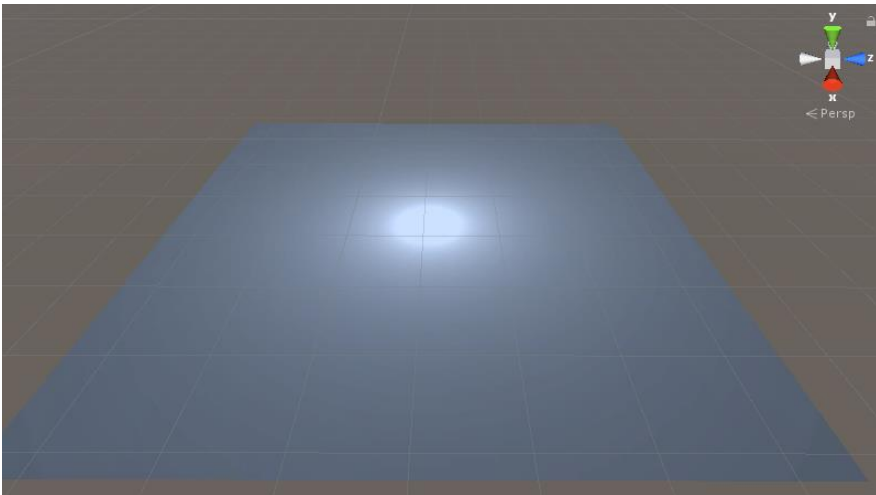


Puissance

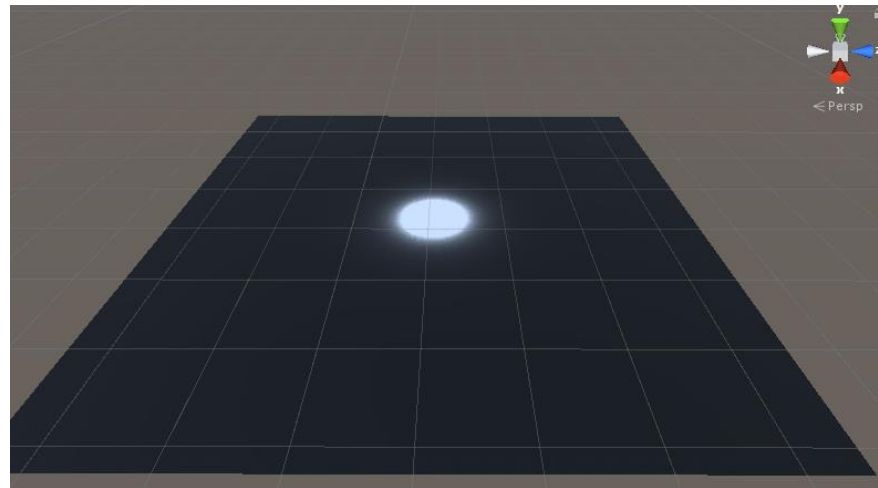


Différence de puissance

Puissance 0,5



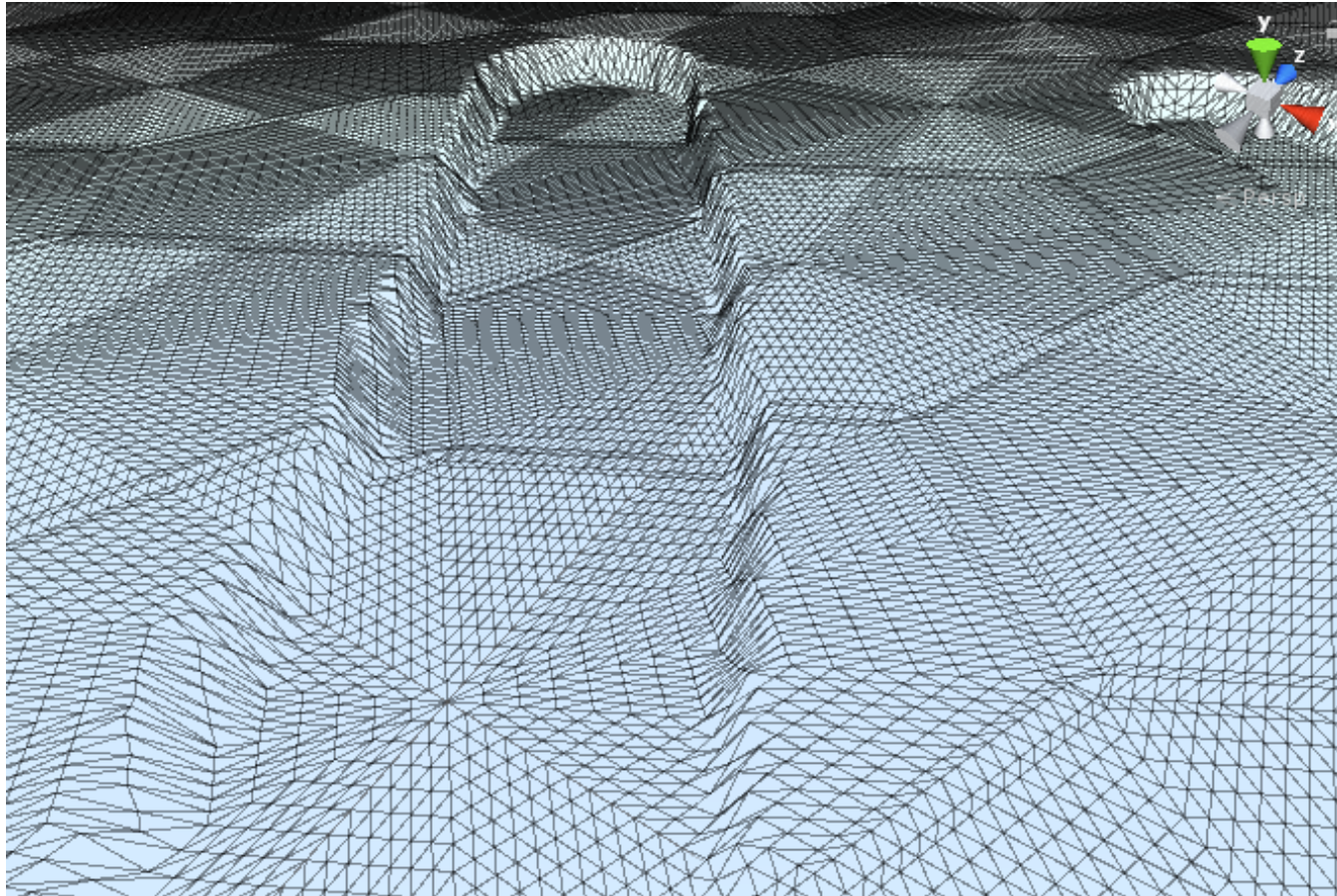
Puissance 4



Déplacement de sommet

Il est possible dans un shader de déplacer les sommets d'une surface. Il est important de comprendre que ceci ne modifie rien d'autre que l'affichage. Les collisions se font toujours au même niveau.

Exemple de déplacement de sommet



Explication du code dans un surface Shader

Shader lab

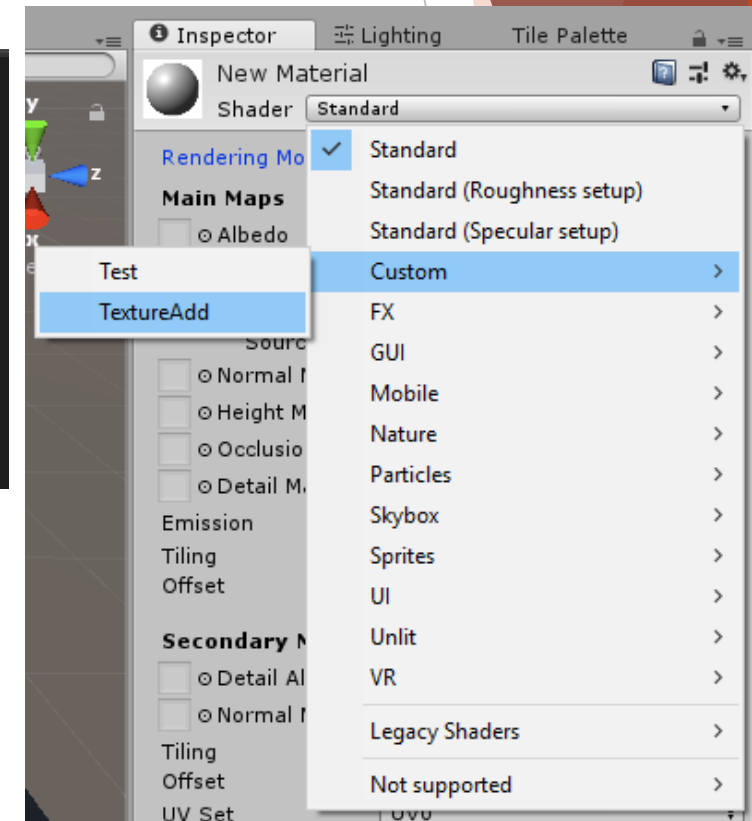
Dans unity, on écrit le shader avec la syntaxe shaderlab. Unity transformera ensuite ce code dans le vrai langage du shader.

L'écriture du shader

Lorsqu'on écrit le code du shader, on l'écrit pour 1 seul pixel. Il faut ensuite s'imaginer que ce code sera exécuté pour chaque pixel auquel ce shader s'applique.

Ajout d'un shader pour les materials

```
Shader "Custom/TextureAdd" {  
    Properties {  
        _Tex1 ("Texture1", 2D) = "white" {}  
        _Tex2 ("Texture2", 2D) = "white" {}  
        _Glossiness ("Smoothness", Range(0,1)) = 0.5  
        _Metallic ("Metallic", Range(0,1)) = 0.0  
    }  
}
```



Ajout de propriété pour un shader

```
Shader "Custom/TextureAdd" {  
    Properties {  
        _Tex1 ("Texture1", 2D) = "white" {}  
        _Tex2 ("Texture2", 2D) = "white" {}  
        _Glossiness ("Smoothness", Range(0,1)) = 0.5  
        _Metallic ("Metallic", Range(0,1)) = 0.0  
    }  
    SubShader {  
        Tags { "RenderType" = "Opaque" }  
        LOD 100  
    }  
}
```

The image shows the Unity 'New Material' inspector for a custom shader named 'Custom/TextureAdd'. The shader code on the left defines two texture properties, _Tex1 and _Tex2, both set to 'white'. It also defines _Glossiness (Smoothness) and _Metallic properties. The inspector on the right shows the corresponding UI elements: two texture slots, Texture1 and Texture2, each with tiling and offset settings. Below these are sliders for Smoothness (set to 0.5) and Metallic (set to 0). At the bottom are checkboxes for Render Queue (set to 'From Shader'), Enable GPU Instancing, and Double Sided Global Illumination. Blue arrows point from the code on the left to the corresponding UI elements: from '_Tex1' to the first texture slot, from '_Tex2' to the second texture slot, from '_Glossiness' to the Smoothness slider, and from '_Metallic' to the Metallic slider.

Définition d'une propriété

```
_Tex1 ("Texture1", 2D) = "white" {}
```

Nom dans
le shader

Nom dans le
material de
Unity

Type de
données

Valeur par
défaut

Début du SubShader

Indique le code du
shader débute

```
Properties {  
    _Tex1 ("Texture1", 2D) = "white" {}  
    _Tex2 ("Texture2", 2D) = "white" {}  
    _Glossiness ("Smoothness", Range(0,1)) = 0.5  
    _Metallic ("Metallic", Range(0,1)) = 0.0  
}  
SubShader{  
    Tags { "RenderType" = "Opaque" }
```

Le tag

```
SubShader{  
  Tags { "RenderType" = "Opaque" }  
  LOD 100  
  
  CGPROGRAM  
  #pragma surface surf Standard fullforwardshadows  
  #pragma target 3.0  
}
```

Indique quand doit
être rendu le
shader.

Level of detail

```
Tags { "RenderType" = "Opaque" }  
LOD 100
```

```
CGPROGRAM  
#pragma surface surf Standard fullforwardshadows  
#pragma target 3.0  
  
sampler2D _Tex1;  
sampler2D _Tex2;
```

Indique le « Level
of detail »

Début du CG

```
Tags { "RenderType" = "Opaque" }  
LOD 100  
  
CGPROGRAM  
#pragma surface surf Standard fullforwardshadows  
#pragma target 3.0  
  
sampler2D _Tex1;  
sampler2D _Tex2;
```

Indique le
début du
programme
CG ou C for
graphics

Instruction de compilation

```
Tags { "RenderType" = "Opaque" }  
LOD 100
```

```
CGPROGRAM
```

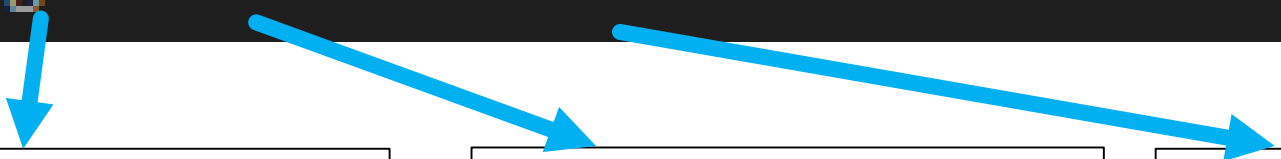
```
#pragma surface surf Standard fullforwardshadows  
#pragma target 3.0
```

```
sampler2D _Tex1;  
sampler2D _Tex2;
```

Instruction
de
compilation

Explication des instructions de compilation

```
#pragma surface surf Standard fullforwardshadows
```



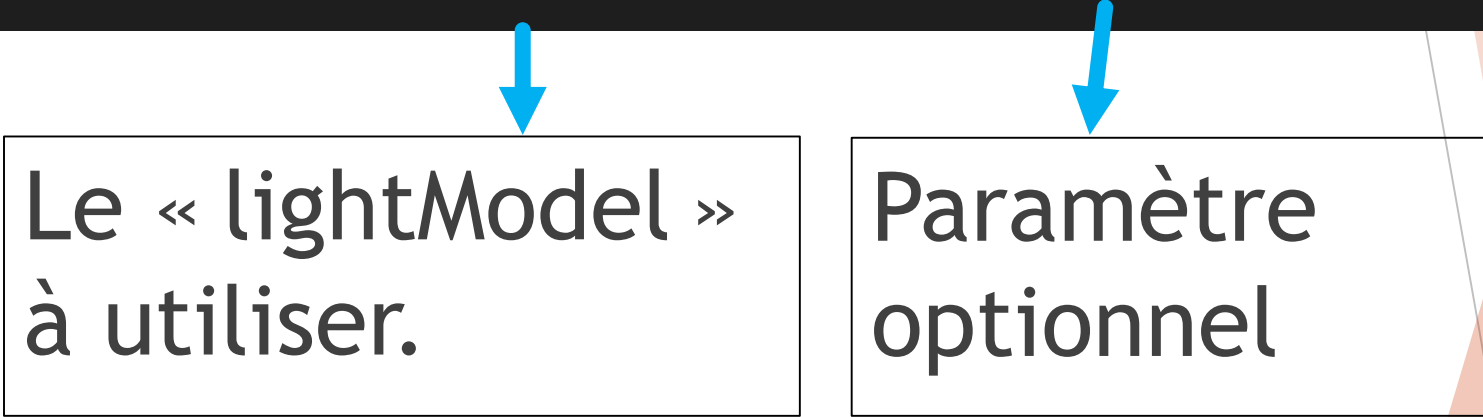
Indique une instruction de compilation

Un shader de surface à besoin d'une fonction de surface

La fonction de surface s'appellera « surf » et sera définie plus tard dans le code CG.

Explication des instructions de compilation

```
#pragma surface surf Standard fullforwardshadows
```



Le « lightModel »
à utiliser.

Paramètre
optionnel


```
#pragma surface surf Standard fullforwardshadows  
#pragma target 3.0
```

```
sampler2D _Tex1;  
sampler2D _Tex2;  
half _Glossiness;  
half _Metallic;
```

```
struct Input {  
    float2 uv_Tex1;  
};
```

Définition de
variable dans
le SubShader.

```
#pragma surface surf Standard fullforwardshadows
#pragma target 3.0

sampler2D _Tex1;
sampler2D _Tex2;
half _Glossiness;
half _Metallic;

struct Input {
    float2 uv_Tex1;
};
```

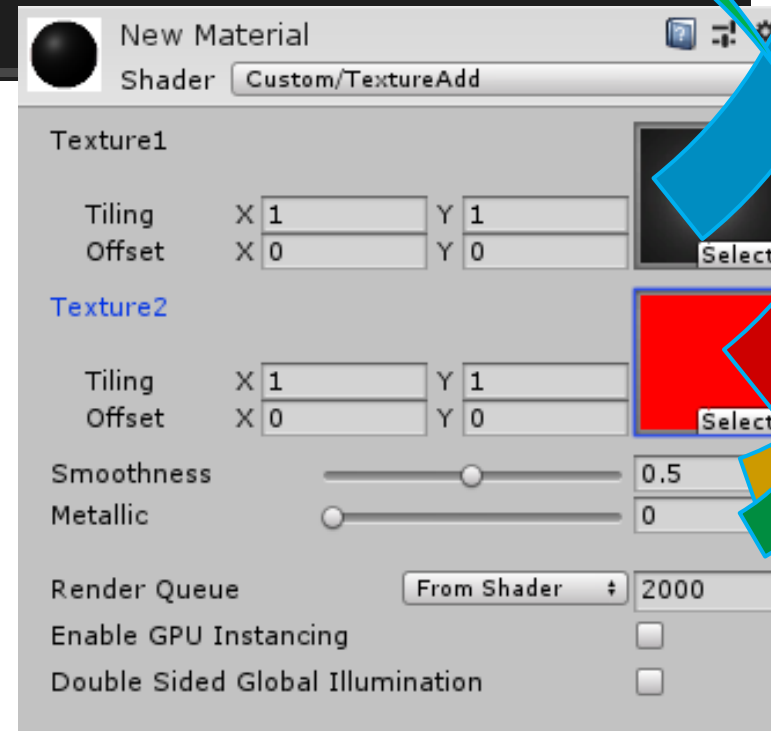
Pour avoir les UVs d'une texture le nom doit être "uv_"+nom de la texture ou "uv2_nom de la texture. Ceci est sensible à la case.

Définition de valeur à partir des propriétés

Si une variable porte le même nom que son nom dans les propriétés, sa valeur sera transférée.

```
Properties {  
  _Tex1 ("Texture1", 2D) = "white" {}  
  _Tex2 ("Texture2", 2D) = "white" {}  
  _Glossiness ("Smoothness", Range(0,1)) = 0.5  
  _Metallic ("Metallic", Range(0,1)) = 0.0  
}
```


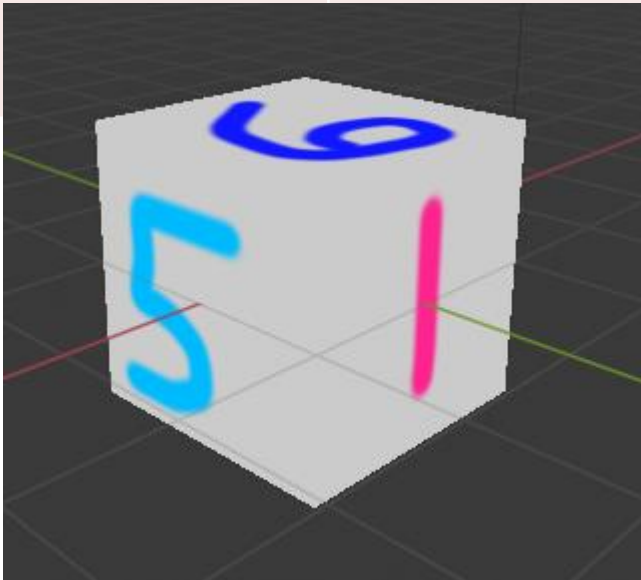
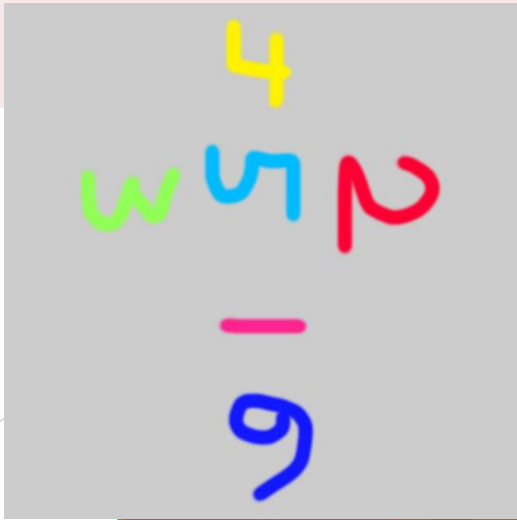
```
sampler2D _Tex1;  
sampler2D _Tex2;  
half _Glossiness;  
half _Metallic;
```



Type de variable importante dans le CG (Chiffre)

Nom	Fonction	Utilisation
Float	Float sur 32 bits	Positions dans le monde, coordonnée de texture et calculation
half	Float sur 16 bits	Vecteur court, directions, couleur dynamique
Fixed	Type signé entre -2 et 2 avec 10 bits de précision (total 1 bits)	Couleur régulière et opération simple sur les couleurs
int	Entier signé sur 32 bits	Compteur, indice de tableau

Type de variable importante dans le CG (Chiffre)

Nom	Fonction	Utilisation
sampler2D sampler2D_half sampler2D_float	Texture 2D	
samplerCUBE samplerCUBE_half samplerCUBE_float	Vecteur avec X dimensions de type float	 

Half pour basse précision et float pour grande précision

Type de variable Array

Nom	Fonction	Utilisation
FloatX	Vecteur avec X dimensions de type float	
Float3x3	(float3, float3, float3) Matrice de 3 colonnes et 3 lignes	Int3x3 matrice; Pour obtenir l'élément colonne 1 ligne 3 : matrice._m20;

int2	float2	half2	fixed2	...
Int3	float3	half3	fixed3	...
...

Définition d'une structure pour les données d'entrée

```
struct Input {  
    float2 uv_Tex1;  
};
```



Les données pour un surface shader (les plus importants)

Type	Nom	Description
Float3	viewDir	Vecteur de direction de la vue
Float4	screenPos	Vecteur de la position sur l'écran
Float3	worldPos	Vecteur de la position dans le monde
Float3	worldNormal	Vecteur de la normal
Float2	uv+[nom de la texture]	Vecteur de la position sur une texture 2D

Définition de la fonction surface

```
struct Input {  
    float2 uv_Tex1;  
};
```

On reçoit dans notre fonction surf les valeurs présentes dans notre structure.



```
void surf(Input IN, inout SurfaceOutputStandard o) {  
    fixed4 c = pow(tex2D(_Tex1, IN.uv_Tex1), 4) * tex2D(_Tex2, IN.uv_Tex1);  
    o.Albedo = c.rgb;  
  
    o.Metallic = _Metallic;  
    o.Smoothness = _Glossiness;  
    o.Alpha = c.a;  
}  
ENDCG
```

Définition de la fonction surface

```
void surf(Input IN, inout SurfaceOutputStandard o) {  
    fixed4 c = pow(tex2D(_Tex1, IN.uv_Tex1),4) * tex2D(_Tex2, IN.uv_Tex1);  
    o.Albedo = c.rgb;  
  
    o.Metallic = _Metallic;  
    o.Smoothness = _Glossiness;  
    o.Alpha = c.a;  
}  
ENDCG
```

Paramètre de sortie. On veut définir les valeurs en sortie.

Définition de la fonction surface

```
void surf(Input IN, inout SurfaceOutputStandard o) {  
    fixed4 c = pow(tex2D(_Tex1, IN.uv_Tex1),4) * tex2D(_Tex2, IN.uv_Tex1);  
    o.Albedo = c.rgb;  
  
    o.Metallic = _Metallic;  
    o.Smoothness = _Glossiness;  
    o.Alpha = c.a;  
}  
ENDCG
```

Contient RGBA : matrice de 4 variables de type fixed

Définition de la fonction surface

```
void surf(Input IN, inout SurfaceOutputStandard o) {  
    fixed4 c = pow(tex2D(_Tex1, IN.uv_Tex1),4) * tex2D(_Tex2, IN.uv_Tex1);  
    o.Albedo = c.rgb;  
  
    o.Metallic = _Metallic;  
    o.Smoothness = _Glossiness;  
    o.Alpha = c.a;  
}  
ENDCG
```

Puissance 4

Définition de la fonction surface

```
void surf(Input IN, inout SurfaceOutputStandard o) {  
    fixed4 c = pow(tex2D(_Tex1, IN.uv_Tex1), 4) * tex2D(_Tex2, IN.uv_Tex1);  
    o.Albedo = c.rgb;  
  
    o.Metallic = _Metallic;  
    o.Smoothness = _Glossiness;  
    o.Alpha = c.a;  
}  
ENDCG
```

Lecture d'une texture à une position :

tex2D (sampler2D texture, float2 position)

Définition de la fonction surface

```
void surf(Input IN, inout SurfaceOutputStandard o) {  
    fixed4 c = pow(tex2D(_Tex1, IN.uv_Tex1),4) * tex2D(_Tex2, IN.uv_Tex1);  
    o.Albedo = c.rgb;  
  
    o.Metallic = _Metallic;  
    o.Smoothness = _Glossiness;  
    o.Alpha = c.a;  
}  
ENDCG
```

Assignation du paramètre en sortie au bon RGB.

Définition de la fonction surface

```
void surf(Input IN, inout SurfaceOutputStandard o) {  
  
    fixed4 c = pow(tex2D(_Tex1, IN.uv_Tex1),4) * tex2D(_Tex2, IN.uv_Tex1);  
    o.Albedo = c.rgb;  
  
    o.Metallic = _Metallic;  
    o.Smoothness = _Glossiness;  
    o.Alpha = c.a;  
}  
ENDCG
```

Assignation du paramètre en sortie au bon Metallic, Glossiness et Alpha. D'autres paramètres de sortie sont disponibles.

Définition de la fonction surface

```
void surf(Input IN, inout SurfaceOutputStandard o) {  
  
    fixed4 c = pow(tex2D(_Tex1, IN.uv_Tex1),4) * tex2D(_Tex2, IN.uv_Tex1);  
    o.Albedo = c.rgb;  
  
    o.Metallic = _Metallic;  
    o.Smoothness = _Glossiness;  
    o.Alpha = c.a;  
}  
ENDCG
```

On termine le code CG.

Définition de la fonction surface

```
}  
Fallback "Diffuse"  
}
```

On ferme les accolades et on indique un « fallback » si notre shader n'est pas compatible avec la carte graphique.