

Mondes Virtuels

Visiter une maquette virtuelle

Eric Maisel

Ecole Nationale d'Ingénieurs de Brest

Printemps 2025

Introduction

Introduction

Réalité Virtuelle

Réalité Virtuelle (I. Thouvenin)

"Faire vivre à un utilisateur une expérience sensorielle dans un **environnement** créé numériquement qui simule la réalité ou un monde imaginaire.

Créant artificiellement des sensations (vision, son, toucher) elle donne un sentiment fort d'immersion et elle permet des interactions."

Monde Virtuel

Monde virtuel : ensemble de modèles (3d ?) qui constitue l'**environnement**

Introduction

Réalité Virtuelle

Immersion

Faire croire qu'on est ailleurs

Autonomie

La Réalité c'est quand on se cogne (Lacan)

Introduction

Pourquoi ?

A des fins de

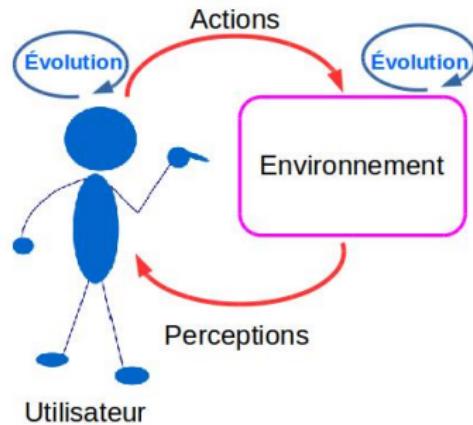
- Conception / Validation
- Transmission / Formation
- Bien être / Loisirs

Dans des mondes

- N'existant pas
- N'existant plus
- Trop lointains
- Trop coûteux
- Trop dangereux

Introduction

Couplage sensori-moteur fort



- **Couplage sensori-moteur** : gestes / perceptions
- **Couplage fort** : temps réel / faible latence

Introduction

Plate-forme d'exécution

Plate-forme native

- Exemple : Unity 3d, Unreal Engine

3d dans un navigateur Web

- Exemple : Three.js, Babylonjs
- Intérêt
 - Facilité d'installation
 - Multi plateforme
 - Accès + naturel à des serveurs (vers la RV distribuée)

Introduction

Niveaux d'abstraction

Niveau matériel (WebGL)

- **Données** : nombres, coordonnées, matrices
- **Structures de données** : registres de couleurs, piles de matrices, files de sommets

Niveau procédural (THREE.js, BABYLONJS)

- **Données** : objets géométriques, groupes d'objets, matériaux, transformations géométriques
- **Structures de données** : graphes de scène

Niveau déclaratif

- **Données** : entités, forces
- **Structures de données** : relations entre entités

Introduction

Objectif

Visiter une maquette virtuelle

- Créer le monde virtuel
- Contrôler le déplacement du visiteur dans le monde virtuel
- Interroger les objets du monde virtuel

Animer les objets virtuels

- Appliquer des forces aux objets du monde virtuel
- Contrôler l'application des forces en vue d'atteindre un objectif
- Utiliser des forces pour animer une foule

Rendre autonome les objets

- Permettre aux objets de percevoir leur environnement
- Permettre la prise de décision en utilisant des automates et des systèmes à base de règles

Infrastructure 3d

Base

- 3d dans un navigateur Web
- Script embarqué dans une page HTML
- Script écrit en JavaScript

Extrait d'une page Web

```
<body>
  <canvas id="renderCanvas"></canvas>
  <script type="module" src="js/main.js"></script>
</body>
```

- **Canvas** : zone d'affichage des images calculées
- **Engine** : procédures de calcul des images de la scène telle que la voit la caméra
- **Scene** : collection d'objets 3d et de sources lumineuses
- **Camera** : description des relations entre l'espace 3d et un plan image

Infrastructure 3d

Création d'un moteur de rendu

```
const canvas = document.getElementById('renderCanvas') ;  
  
const engine = new BABYLON.Engine(canvas, true) ;  
  
...
```

Infrastructure 3d

Création de la scène

...

```
const createScene = function(canvas, engine){  
    const scn = new BABYLON.Scene(engine) ;  
    // ICI : création du contenu de la scène  
    return scn ;  
}
```

```
const scene = createScene(canvas, engine) ; ..
```

Infrastructure 3d

Création de caméra

Création de la caméra

```
const pos = new BABYLON.Vector3(5, 1.7, 5);  
const camera = new BABYLON.UniversalCamera("cam", pos, scene);
```

Orientation de la caméra

```
camera.setTarget(new BABYLON.Vector3(0, 1.5, 0));
```

Contrôle interactif de la caméra

```
camera.attachControl(canvas) ;
```

Infrastructure 3d

Affichage d'images

Affichage d'une image

```
scene.render() ;
```

Affichage d'une séquence d'images

```
engine.runRenderLoop(() => {scene.render() ;}) ;
```

Infrastructure 3d

Création du temps



t : horloge du monde physique - τ : horloge du monde virtuel

Echantillonnage temporel

- Image I_k calculée/affichée en t_k
- $\Delta t_{k,k+1} = t_{k+1} - t_k$

Egalité des écoulements temporels

- $\tau_0 = t_0$
- $\tau_{k+1} = \tau_k + \Delta t_{k,k+1}$

Infrastructure 3d

Création du temps

```
var tau = 0 ;  
  
engine.runRenderLoop(() => {  
    const dt = engine.getDeltaTime()/1000.0 ;  
    tau = tau + dt ;  
    // ICI : code pour faire évoluer l'état du monde virtuel  
    scene.render() ;  
})
```

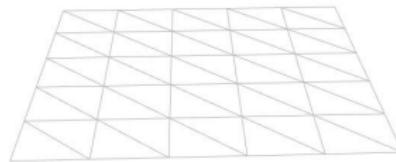
Objets

Mesh

- Objets : représentés par leurs surfaces
- Surface = Géométrie + Matériaux
- Géométrie représentée par un **Mesh** :
 - sommets,
 - arêtes
 - faces

Objets

Un sol



```
const params = {width:10, height:10, subdivisions:5} ;  
const sol = BABYLON.MeshBuilder.CreateGround("sol",  
                                         params,  
                                         scene) ;
```

Objets

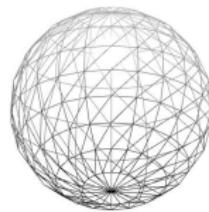
Une boîte



```
const params = {width:3, height:0.5, depth:1} ;  
const sol = BABYLON.MeshBuilder.CreateBox("boite",  
                                         params,  
                                         scene) ;
```

Objets

Une sphère



```
const params = {diameter:2, segments:8} ;  
const sol = BABYLON.MeshBuilder.CreateSphere("sphere",  
                                              params,  
                                              scene) ;
```

Objets

Un rectangle

```
const params = {width:3, height:2} ;  
const rect    = BABYLON.MeshBuilder("rect1", params, scene) ;
```

Objets

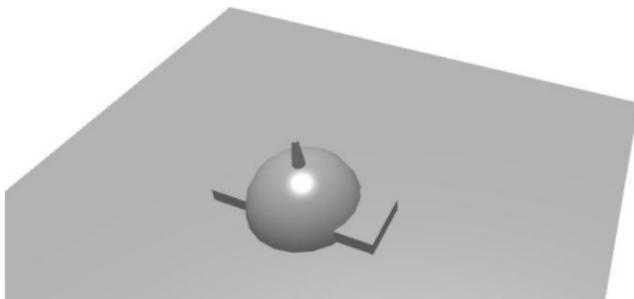
Fonctions de création d'une primitive 3d

BABYLON.MeshBuilder.CreateXX(nom, params, scene)

- Dans le module BABYLON.MeshBuilder
- CreateXX : fonction décrivant un objet XX
- Paramètres params décrivant :
 - la forme de l'objet
 - la précision de l'approximation polyédrique
 - la taille de l'objet

Objets

Fonctions de création d'une primitive 3d



```
const sol = BABYLON.MeshBuilder.CreateGround(...) ;  
const sph = BABYLON.MeshBuilder.CreateSphere(...) ;  
const cyl = BABYLON.MeshBuilder.CreateCylinder(...) ;  
const box = BABYLON.MeshBuilder.CreateBox(...) ;
```

Objets centrés sur l'origine du repère du monde virtuel

Transformations géométriques

Objectif

- Constat : les objets primitifs sont centrés sur l'origine du repère du monde
- Objet : Placer les objets dans l'espace

Transformations géométriques

Transformations géométriques élémentaires

Transformations affines élémentaires

- **Translation**

```
mesh.position = new BABYLON.Vector3(2,1,3);
```

- **Rotation**

```
mesh.rotation = new BABYLON.Vector3(0,Math.PI/3,0);
```

- **Mise à l'échelle**

```
mesh.scaling = new BABYLON.Vector3(1.2,1,1.2);
```

Composition de transformations affines

- **TRS**

- Utilisation de graphe de scène

Transformations géométriques

Graphe de scène

Exemple : accrocher 2 tableaux à une cloison



Objets et repères

- Scene : repère R_0
- Cloison : repère R_1
- Tableau A : repère R_2
- Tableau B : repère R_3

Transformations géométriques

Graphe de scène

- P un point de la cloison
- A la modélisation est donné : P/R_1
- A la visualisation est utilisé : $P/R_0 = T_1^0 P/R_1$
- T_1^0 :
 - Expression de P de R_1 dans R_0
 - Placement de R_1 par rapport à R_0

Transformations géométriques

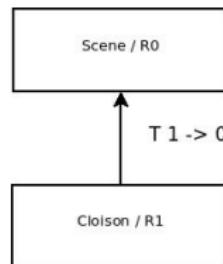
Graphe de scène

Exemple : accrocher 2 tableaux à une cloison

```
cloison = ...
```

```
cloison.position = new BABYLON.Vector3(5,1.5,5) ;
```

```
cloison.rotation = new BABYLON.Vector3(0, Math.PI/3,0) ;
```



Transformations géométriques

Graphe de scène

- P un point du tableau A
- A la modélisation est donné : P/R_2
- A la visualisation est utilisé : $P/R_0 = T_1^0 P/R_1 = T_1^0 T_2^1 P/R_2$
- T_j^i :
 - Expression de P de R_j dans R_i
 - Placement de R_j par rapport à R_i

Transformations géométriques

Graphe de scène : application dans BABYLONJS

```
cloison = ...
```

```
cloison.position = new BABYLON.Vector3(5,1.5,5) ;
```

```
cloison.rotation = new BABYLON.Vector3(0, Math.PI/3,0) ;
```

```
tableauA = ...
```

```
tableauA.parent = cloison ;
```

```
tableauA.position = new BABYLON.Vector3(2,2,0.15);
```

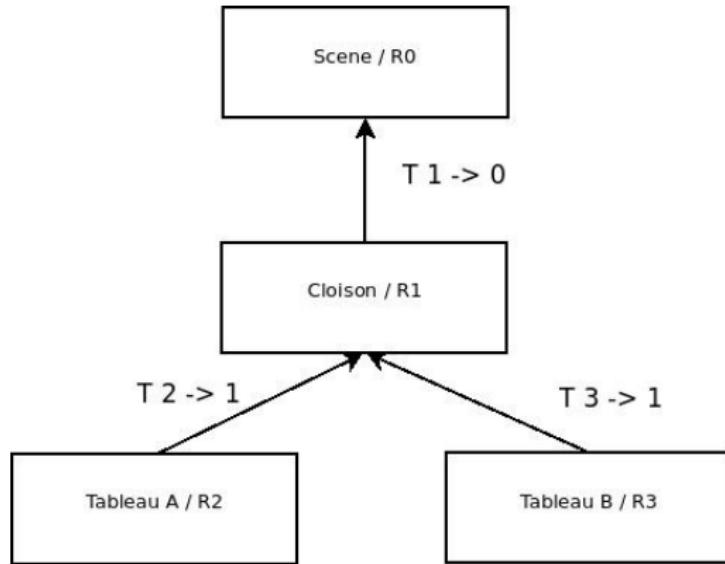
```
tableauB = ...
```

```
tableauB.parent = cloison ;
```

```
tableauB = new BABYLON.Vector3(-2.5, 2, 0.15) ;
```

Transformations géométriques

Graphe de scène : application dans BABYLONJS



Transformations géométriques

Graphe de scène : application dans BABYLONJS

Utilisation d'un TransformNode

- Objet non rendu utilisé comme centre de transformation
- utilisable comme un Mesh uniquement pour le placement des objets

```
function Cloison(nom,data, scene){  
  
    const h = data.hauteur || 3.0 ;  
    const l = data.largeur || 5.0 ;  
    const e = data.epaisseur || 0.1 ;  
  
    const groupe = new TransformNode("tf-"+nom, scene) ;  
  
    ...  
  
    groupe.position.x = -l/2 ;  
    groupe.position.y = e/2 ;  
    groupe.position.z = h/2 ;  
  
    ...  
  
    groupe.rotation.x = 0 ;  
    groupe.rotation.y = 0 ;  
    groupe.rotation.z = 0 ;  
  
    ...  
  
    return groupe ;  
}
```

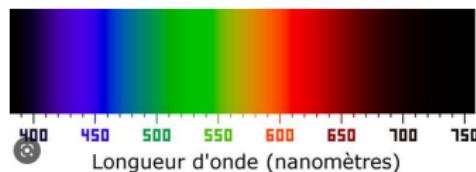
Transformations géométriques

Graphe de scène : application dans BABYLONJS

```
...  
const opt = {width:l, height:h, depth:e};  
const boite  = BABYLON.MeshBuilder.CreateBox(  
    nom,  
    opts,  
    scene);  
boite.parent = groupe ;  
boite.position = new BABYLON.Vector3(0,h/2,0) ;  
  
return groupe ;  
}
```

Eclairage

La lumière



Phénomène naturel

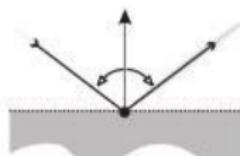
- Superposition de rayonnements E.M. élémentaires
- Oeil insensible à la phase
- Représentation par un spectre : $\Phi(\lambda)$

Représentation numérique

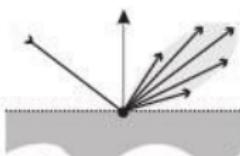
- Simulation d'éclairage : 20 échantillons / spectre
- Infographie Temps Réel : 3 échantillons / spectre

Eclairage

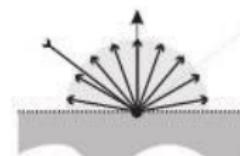
Propagation de la lumière



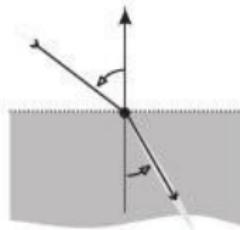
(a) Réflexion parfaite



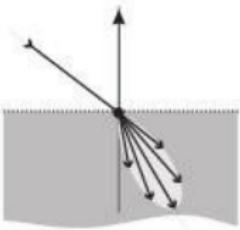
(b) Réflexion spéculaire



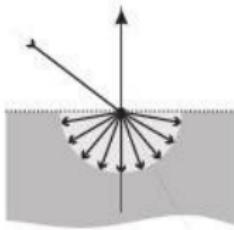
(c) Réflexion diffuse



(d) Réfraction parfaite



(e) Réfraction spéculaire

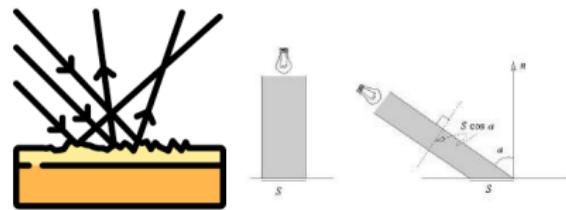


(f) Réfraction diffuse

Eclairage

Matériaux Lambertiens

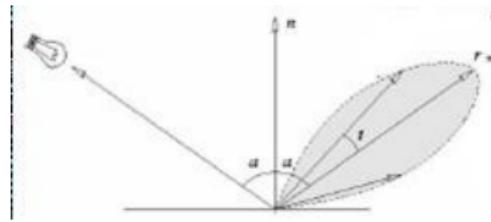
La lumière est réfléchie de façon uniforme dans toutes les directions



$$I_r(\lambda) = I(\lambda)k_d(\lambda) \cos(\vec{N}, \vec{L}) = I(\lambda)k_d(\lambda)$$

Eclairage

Matériaux spéculaires



$$I_s(\lambda) = I(\lambda) K_s \cos^p(t) = I(\lambda) K_s \max\left(\frac{(\vec{R} \cdot \vec{V})^p}{||\vec{R}|| ||\vec{V}||}, 0\right)$$

Eclairage

Sources lumineuses directionnelles

Emet des flux parallèles

Ex// une source très lointaine (le soleil, ...)

```
const light = new BABYLONDirectionalLight(  
    "light0",                                // Nom  
    new BABYLONVector3(1,-1,0),      // Direction  
    scene  
)
```

Eclairage

Sources lumineuses ponctuelles

Emet des flux dans toutes les directions à partir d'un point.
Ex// une ampoule.

```
const light = new BABYLON.PointLight(  
    "light1", // Nom  
    new BABYLON.Vector3(0,3,0), // Position  
    scene  
)
```

Eclairage

Sources lumineuses "spot"

Emet des flux dans un angle solide à partir d'un point.

```
const spot = BABYLON.SpotLight(  
    "spot0",  
    position, direction,  
    angle_spread, speed_of_dissipation  
) ;
```

```
spot.diffuse = BABYLON.Color3.Yellow() ;
```

Eclairage

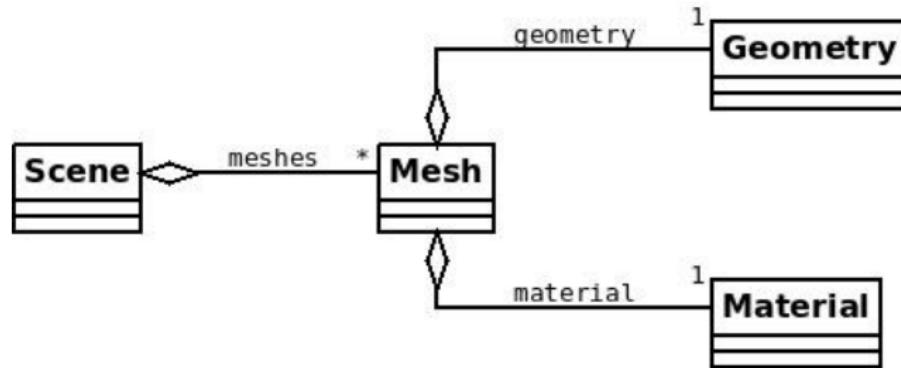
Sources lumineuses ambiantes

- Simuler un éclairage ambiant
- up : dirigé vers le haut

```
const up      = new BABYLON.Vector3(0,1,0);
const light = new BABYLON.HemisphericLight("10", up, scene);
light.intensity = 0.2 ;
```

Eclairage

Matériau standard



Création d'un matériau

```
const mesh = new BABYLON.MeshBuilder.CreateSphere(...);  
const material = new BABYLON.StandardMaterial("mat", scene);  
mesh.material = material ;
```

Eclairage

Matériau standard : composante diffuse



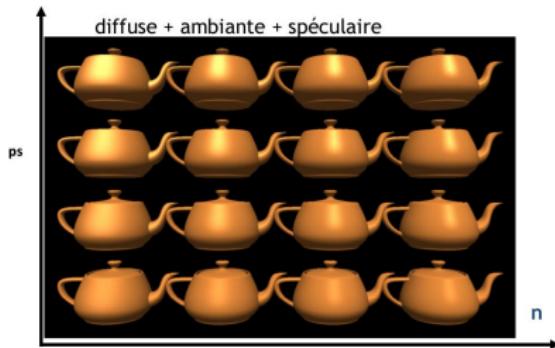
On augmente **pd**, **pa** = 0

Spécification de la composante diffuse

```
mesh.material.diffuseColor = new BABYLON.Color3(0, 1, 0);
```

Eclairage

Matériaux standard : composante spéculaire



Spécification de la composante spéculaire

```
mesh.material =  
    new BABYLON.StandardMaterial("mat_spec", scene) ;  
mesh.material.specularColor = new BABYLON.Color3(1, 0, 0);  
mesh.material.shininess = 30 ;
```

Eclairage

Matériau standard : composante ambiante



On augmente **pa**

Spécification de la composante ambiente

```
mesh.material.ambientColor = new BABYLON.Color3(0, 1, 0);
```

Eclairage

Matériaux standard : composante d'auto-émission

Problème : visualiser une source lumineuse (Ex : une ampoule)

- Les sources lumineuses : métaphore math sans incarnation
- Solution :
 - Source lumineuse ponctuelle en P_0
 - Incarnation : sphère centrée en P_0
 - Contrainte : pas d'effets des autres sources lumineuses

Spécification de la composante d'auto-émission

```
const light = new BABYLON.PointLight( ... , ) ;  
const ampoule = BABYLON.MeshBuilder.CreateSphere( ... ) ;  
ampoule.material = new BABYLON.StandardMaterial( ... ) ;  
ampoule.material.emissiveColor = BABYLON.Color3.Yellow() ;
```

Aspect des objets

Modèle additif (sources, fréquences, modèles d'éclairement)

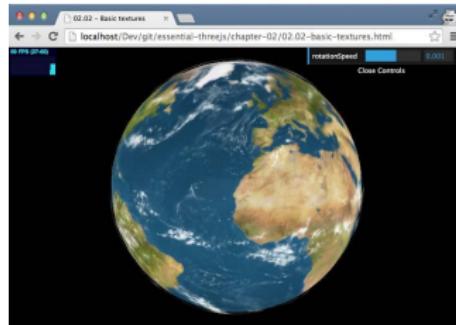
$$I_p(\lambda) = p_a I_a(\lambda) + p_d(\lambda) \cos(\theta) I_s(\lambda) + p_s \cos^n(\theta') I_s(\lambda)$$



Textures

Textures de couleurs

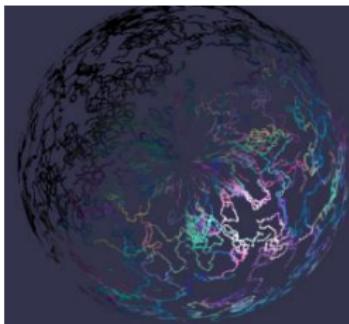
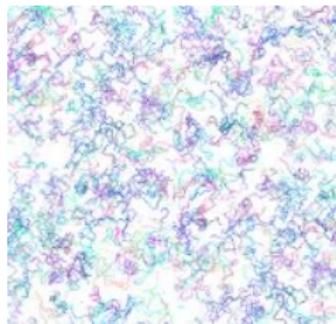
- Description de variations de couleurs à la surface d'un objet
- Description au moyen d'une image



```
const mat = new BABYLON.StandardMaterial("mat", scene);
mat.diffuseTexture = new BABYLON.Texture("ima.jpg", scene);
mesh.material = mat ;
```

Textures

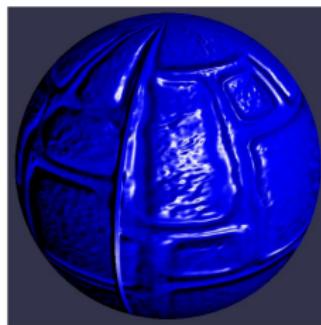
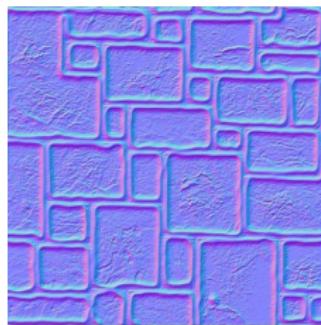
Textures de couleurs avec transparence



```
const mat = new BABYLON.StandardMaterial("mat", scene) ;  
mat.diffuseTexture = new BABYLON.Texture("ima.png", scene) ;  
mat.diffuseTexture.hasAlpha = true ;
```

Textures

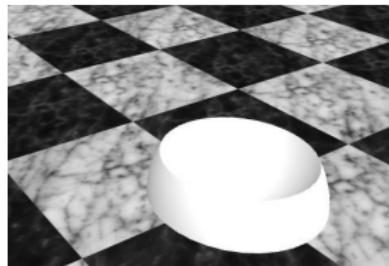
Textures de relief



```
const mat = new BABYLON.StandardMaterial("mat", scene);
mat.bumpTexture = new BABYLON.Texture(cheminAcces, scene);
```

Représentation d'objets 3d

Objets manufacturés



- Une écuelle : forme "complexe"

```
const sph0 = creerSphere(scene) ;  
const sph1 = creerSphere(scene) ;  
sph1.position.y = 0.5 ;
```

Représentation d'objets 3d

Objets manufacturés

```
const csg0 = BABYLON.CSG.FromMesh(sph0);
const csg1 = BABYLON.CSG.FromMesh(sph1) ;
csg0.subtractInPlace(csg1);
const mesh = csg0.toMesh() ;

sph0.dispose() ;
sph1.dispose() ;
```

Représentation d'objets 3d

Objets manufacturés

- Objets solides de points
- Opérateurs ensemblistes :
 - Intersection
 - Union
 - Différence

Comment faire des fenêtres, des portes ?

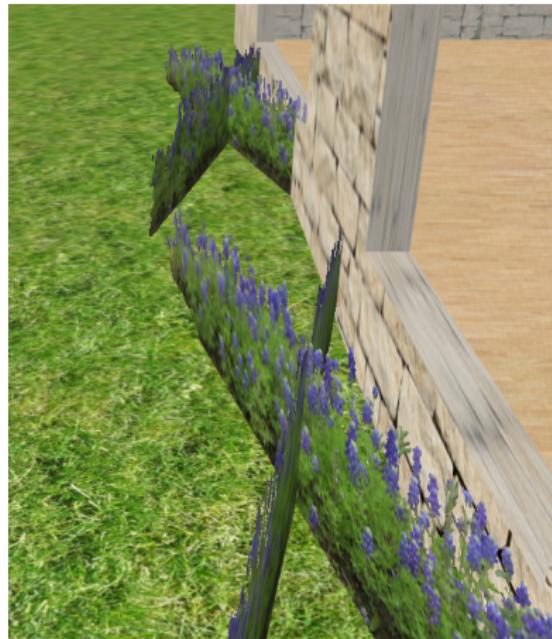
Végétation

Représentation par des plans texturés



Végétation

Représentation par des plans texturés



Végétation

Représentation par des plans texturés

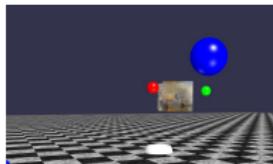


- Objets représentables par leur silhouette
- Objets 3d distants
- Objets 3d "interactables"

Environnement

Arrière-plan

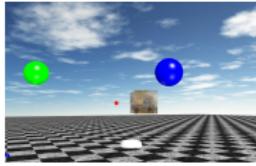
Sans arrière -plan



Sans arrière-plan



Avec arrière-plan



Environnement

Arrière-plan

```
var skybox = BABYLON.Mesh.CreateBox("skyBox", 100.0, scene);
var skyboxMaterial=new BABYLON.StandardMaterial("sbm",scene);
skyboxMaterial.backFaceCulling= false ;
skybox.material=skyboxMaterial ;
skyboxMaterial.diffuseColor = new BABYLON.Color3(0,0,0);
skyboxMaterial.specularColor = new BABYLON.Color3(0,0,0);
skyboxMaterial.reflectionTexture =
    new BABYLON.CubeTexture("skybox/skybox", scene);
skyboxMaterial.reflectionTexture.coordinatesMode =
BABYLON.Texture.SKYBOX_MODE;
```

Environnement

Brouillard

Brouillard linéaire

```
scene.fogMode = BABYLON.Scene.FOGMODE_LINEAR ;  
scene.fogColor = new BABYLON.Color3(0,0,0.01) ;  
scene.fogStart = 20.0 ;  
scene.fogEnd = 80.0 ;
```

Brouillard exponentiel

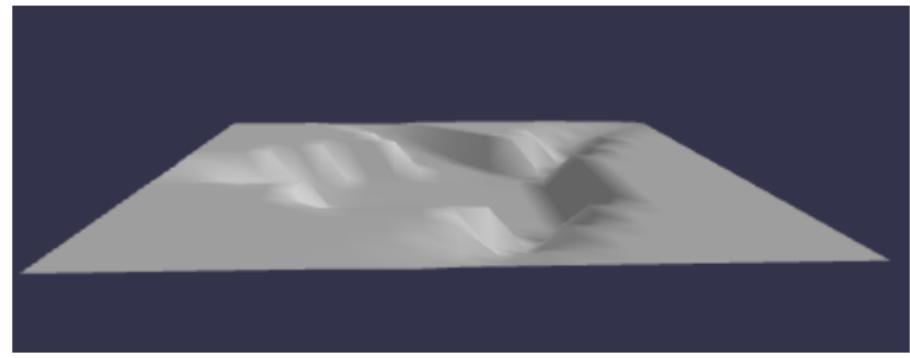
```
scene.fogMode = BABYLON.Scene.FOGMODE_EXP ;  
scene.fogColor = new BABYLON.Color3(0,0,0.01) ;  
scene.fogDensity = 0.02 ;
```

Types de brouillard

FOGMODE_NONE, FOGMODE_LINEAR, FOGMODE_EXP, FOGMODE_EXP2

Environnement

Modèle de terrain



Environnement

Modèle de terrain



Altitudes normalisées

- Noir (0) : altitude 0
- Blanc (255) : altitude 1

$$h(i\Delta_x, j\Delta_y) = h_{min} + (h_{max} - h_{min}) Ima_{i,j}$$

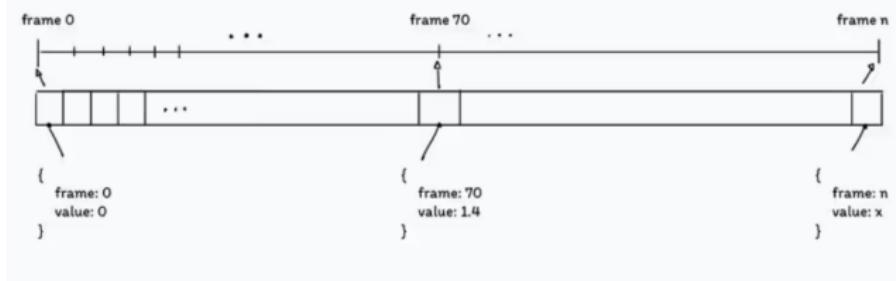
Environnement

Modèle de terrain

Création de la géométrie

```
const hm = BABYLON.MeshBuilder(  
    "terrain", // Nom du terrain,  
    "assets/heightmap/terrain.png", // accès MNT  
    {  
        width:150, height:150,  
        subdivisions: 20,  
        minHeight:0, maxHeight: 4  
    },  
    scene  
);
```

Animation



```
const animation = new BABYLON.Animation(
    'yrot', // nom
    'rotation.y', // propriété à animer
    30, // frames par seconde
    BABYLON.ANIMATIONTYPE_FLOAT, // Type des valeurs
    BABYLONLOOPMODE_CYCLE) // Animation cyclique
```

Animation

```
const animationKeys = [{frame:0, value:0},  
                      {frame:60, value:Math.PI/2}]]  
  
animation.setKeys(animationKeys) ;  
mesh.animations = [] ;  
mesh.animations.push(animation) ;  
  
scene.beginAnimation(  
    mesh, // objet à animer  
    0,    // rang du premier frame  
    60,   // nombre de frames  
    true // true pour cycle  
)
```

Interactions

Interactions

Interagir au moyen du clavier

Directement

```
window.addEventListener('keydown', (e) => {  
    const c = e.keyCode ;  
    if(c == 37){...} else  
    if(c == 39){...} else  
    {...}  
})
```

Interactions

Interagir au moyen du clavier

Indirectement

```
const keys = {"g":False, "d":False, "av":False};  
window.addEventListener('keydown', (e) => {  
    const c = e.keyCode ;  
    if(c == 37){keys.g=True} else  
    if(c == 39){keys.d=True} else  
    {...}  
});
```

```
window.addEventListener('keydown', (e) => {  
    const c = e.keyCode ;  
    if(c == 37){keys.g=True} else  
    if(c == 39){keys.d=True} else  
    {...}  
});
```

Interactions

Interagir au moyen de la souris

Objectif

Sélectionner un objet avec la souris

```
const ray = camera.getForwardRay() ;  
const hit = scene.pickWithRay(ray) ;  
if(hit.pickedmesh){ ... }
```

- `hit.pickedMesh` : référence sur le mesh intersecté le plus proche de la caméra
- `hit.pickedPoint` : Coordonnées (`Vector3`) du point intersecté le plus proche de la caméra
- `hit.distance` : distance de la caméra au point intersecté le plus proche

Interactions

Interagir au moyen de la souris

Sélectionner en désignant avec la souris

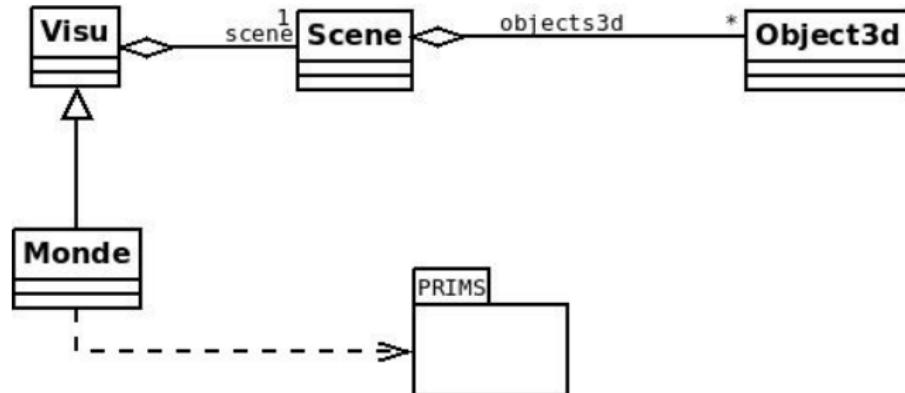
```
window.addEventListener("click", (event)=> {  
    const x = event.x ;  
    const y = event.y ;  
    const pickResult = scene.pick(x,y) ;  
    ...  
})
```

Sélectionner en regardant régulièrement

```
setInterval(()=>{  
    const ray = camera.getForwardRay();  
    const hit = scene.pickWithRay(ray);  
    ...  
}, 10) ;
```

Structuration / Spécification

Visualiseur



- **Visu** : création des images / gestion des assets
- **Monde** : création des objets 3d
- **PRIMS** : catalogue de fonctions de création d'objets 3d

Structuration / Spécification

Visualiseur

```
import {PRIMS} from './prims.js' ;
class Visu {
    constructor(){
        this.canvas = document.getElementById(...);
        this.engine = new BABYLON.Engine(this.canvas, true) ;
        this.scene  = ... ;
        this.camera = ... ;
        this.tau     = 0.0 ;
        ...
    }
}
```

Structuration / Spécification

Visualiseur

```
class Visu {  
    constructor(){  
        ...  
        this.assets = {};  
        ...  
    }  
  
    registerAsset(foo, data){  
        this.assets[name] = foo(name,data, this.scene);  
    }  
  
    findAsset(name){return this.assets[name] || null;}  
    ...  
}
```

Structuration / Spécification

Fonctions de création de "briques" 3d

- Briques de base : caméra, objets 3d, sources lumineuses, matériaux,
...
- Création par des fonctions standardisées :

```
nomBrique(nom,data,scene)
```

```
function creerSphere(name, params, scn){  
    const d      = params.diametre || 1.0;  
    const mat    = params.materiau ;  
    const sph = BABYLON.MeshBuilder.CreateSphere(  
        nom,  
        {diameter:d}, scn);  
    sph.material = mat;  
    return sph;  
}
```

Structuration / Spécification

Fonctions de création de "briques" 3d

```
// Fichier prims.js

function sphere(nom, params, scn){...}

const PRIMS = {
    sphere : creerSphere,
    ...
}

export {PRIMS}

import {PRIMS} from './prims.js' ;

const sph = PRIMS.sphere("sph0", {materiau:{}}, scene)
```

Structuration / Spécification

Monde : création des objets du monde virtuel

```
import {PRIMS} from './prims.js' ;
class Monde extends Visu {

    constructor(){
        super() ;
    }

    genese(){
        // ICI : code de création des objets 3d
        ...
    }
}
export {Monde}
```

Structuration / Spécification

Monde : création des objets du monde virtuel

```
class Monde extends Visu {  
    ...  
  
    genese(){  
        const mesh = PRIMS.sphere(...) ;  
        mesh.position.y = 2 ;  
        ...  
    }  
}
```