

Manuel du TP1

Ce manuel contient une documentation sur le fonctionnement du code ainsi qu'un manuel utilisateur à la fin du document.

Objectif du TP :

- Se familiariser avec WEBGL
- Tracer un cercle

$$\begin{cases} x(t) = 2\cos(t) \\ y(t) = 2\sin(t) \end{cases}$$

- Tracer une courbe paramétrique (un cœur)

$$\begin{cases} x(t) = \sin^3(t) \\ y(t) = \cos(t) - \cos^4(t) \end{cases} \quad \text{avec } t \in [0, 2\pi]$$

- Permettre à l'utilisateur d'afficher des courbes génériques (bonus)
- Permettre à l'utilisateur de tracer ses propres courbes paramétriques (bonus)

Fonctionnement du programme :

Le travail utilise la librairie **Three.JS** pour le rendu graphique

JavaScript :

Nous utilisons threeJS pour gérer la partie graphique. Nous initialisons donc toutes les constantes qui vont nous servir à créer la scène, la caméra, le moteur de rendu et le format des points tracés

```
const scene = new THREE.Scene();
const camera = new THREE.PerspectiveCamera( 50, window.innerWidth / window.innerHeight, 0.1, 1000 );

const renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
document.body.appendChild( renderer.domElement ); //création de la scène

const material = new THREE.PointsMaterial({ //définition taille et couleurs des points
  size: 0.05,
  color: 0x7CFC00
});
```

Ensuite nous déclarons nos deux seules variables globales qui serviront plus tard à stocker la courbe paramétrique donnée par l'utilisateur.

```
var x1 = 0;
var y1 = 0;
```

Pour initialiser la scène avec le point de vue caméra :

```
camera.position.y = -0.5;
camera.position.z = 7.5;
renderer.render( scene, camera );
```

Afin de tracer les différentes courbes, l'appui des boutons html va lancer les fonctions traçant les différentes courbes.

Pour tracer les courbes nous utilisons la fonction de THREE ParametricGeometry (en effet cela semble plutôt cohérent avec le sujet).

L'objet renvoyé par cette fonction est lié à une « texture » pour devenir un mesh que l'on peut ajouter à la scène.

Pour finir il suffit de recharger la scène avec son nouveau contenu.

Le cercle :

```
let drawCircle = () => {
  const geom = new THREE.ParametricGeometry(circleFunction, 500, 1000);
  const circle = new THREE.Mesh( geom, material );
  scene.add( circle ); // on l'ajoute à la scène
  renderer.render( scene, camera );
}
```

Le cœur :

```
let drawHeart = () => {  
  const geometry = new THREE.ParametricGeometry(paramFunction, 500, 1000);  
  const coeur = new THREE.Mesh( geometry, material );  
  scene.add( coeur ); // on l'ajoute à la scène  
  renderer.render( scene, camera );  
}
```

La courbe utilisateur :

```
let drawCurve = () => {  
  x1 = (math.parse(document.getElementById("functionx").value)).compile();  
  y1 = (math.parse(document.getElementById("functiony").value)).compile();  
  console.log(x1);  
  const geoms = new THREE.ParametricGeometry(personalizedParamFunction, 500, 1000);  
  const shape = new THREE.Mesh( geoms, material );  
  scene.add( shape ); // on l'ajoute à la scènes  
  renderer.render( scene, camera );  
}
```

Cette fonction est différente. On voit que le contenu des éléments contenant le texte donné par l'utilisateur est récupéré grâce à la fonction `getElementById()`.

Pour permettre à three d'interpréter ces fonctions nous utilisons la librairie `mathjs` (différent des fonctions `Math`. disponible dans le js). Celle-ci nous permet tout d'abord d'analyser le texte donné (`math.parse`) puis de le compiler à fin qu'il soit évaluable dans le code.

On voit également enfin l'attribution de ces fonctions à nos variables globales. Cela est tout simplement pour des raisons d'optimisation. Il était effectivement possible de le faire dans la fonction `personalizedParamFunction()`, sauf que cela poussera notre navigateur à faire l'opération des milliers de fois au lieu d'une fois.

Ensuite, une à une nous déclarons les fonctions d'évaluations de nos courbes paramétriques que vous avez vu précédemment dans les fonctions.

u est l'ensemble de définition, ici de 0 à 2π

z est la profondeur du point, en effet `threejs` trace en 3D, cela veut donc dire que les points sont en fait des cylindres mais que l'on voit sur le plan de face.

x et y sont bien sur les composantes de la courbe paramétrique

Le cercle :

```
let circleFunction = function (u, v, target) {  
  var u = (u * 2 * Math.PI);  
  
  let x = 2*Math.sin(u);  
  let y = 2*Math.cos(u);  
  let z = v*0.1;  
  
  target.set(x,y,z);  
}
```

Le cœur :

```
let paramFunction = function (u, v, target) {  
  var u = (u * 2 * Math.PI);  
  
  let x = Math.pow(Math.sin(u),3);  
  let y = Math.cos(u) - Math.pow(Math.cos(u),4);  
  let z = v*0.1;  
  
  target.set(x,y,z);  
}
```

La courbe utilisateur :

```
let personalizedParamFunction = function (w, v, target) {  
  var w = (w * 2 * Math.PI);  
  let x = x1.evaluate({u:w});  
  let y = y1.evaluate({u:w});  
  let z = v*0.1;  
  
  target.set(x,y,z);  
}
```

Comme nous l'avons évoqué précédemment les variables x1 et y1 sont prêtes à être évaluées et c'est ce que nous faisons pour chaque points à l'aide de .evaluate() (la variable u prend la valeur w demandée par personalizedParamFunction())

HTML :

Le code HTML utilisé est très simpliste, il se résume à un seul fichier index.html (on observe entre autre l'importation de math.min.js, la librairie utilisée pour extraire la fonction paramétrique donnée par l'utilisateur.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>TP 1</title>
    <link rel="stylesheet" href="style.css">
    <script src="https://cdnjs.cloudflare.com/ajax/libs/mathjs/9.5.0/math.min.js"></script>
  </head>
  <body>
    <div id="menu">
      <div class="options">
        x(u)=<input type="text" id="functionx" /> <!-- composante x de la courbe paramétrique -->
      </div>
      <div class="options">
        y(u)=<input type="text" id="functiony" /><!-- composante y de la courbe paramétrique -->
      </div>
      <div class="options">
        <input type = "button" onclick = "drawCurve()" value = "Tracer">
      </div>
      <div class="options">
        <p>_____</p>
        <p>Courbes paramétriques pré-enregistrées</p>
        <input type = "button" onclick = "drawCircle()" value = "Tracer le cercle">
      </div>
      <div class="options">
        <input type = "button" onclick = "drawHeart()" value = "Tracer le coeur">
      </div>
    </div>
    <script src="three.js"></script>
    <script src="drawing.js"></script>
  </body>
</html>
```

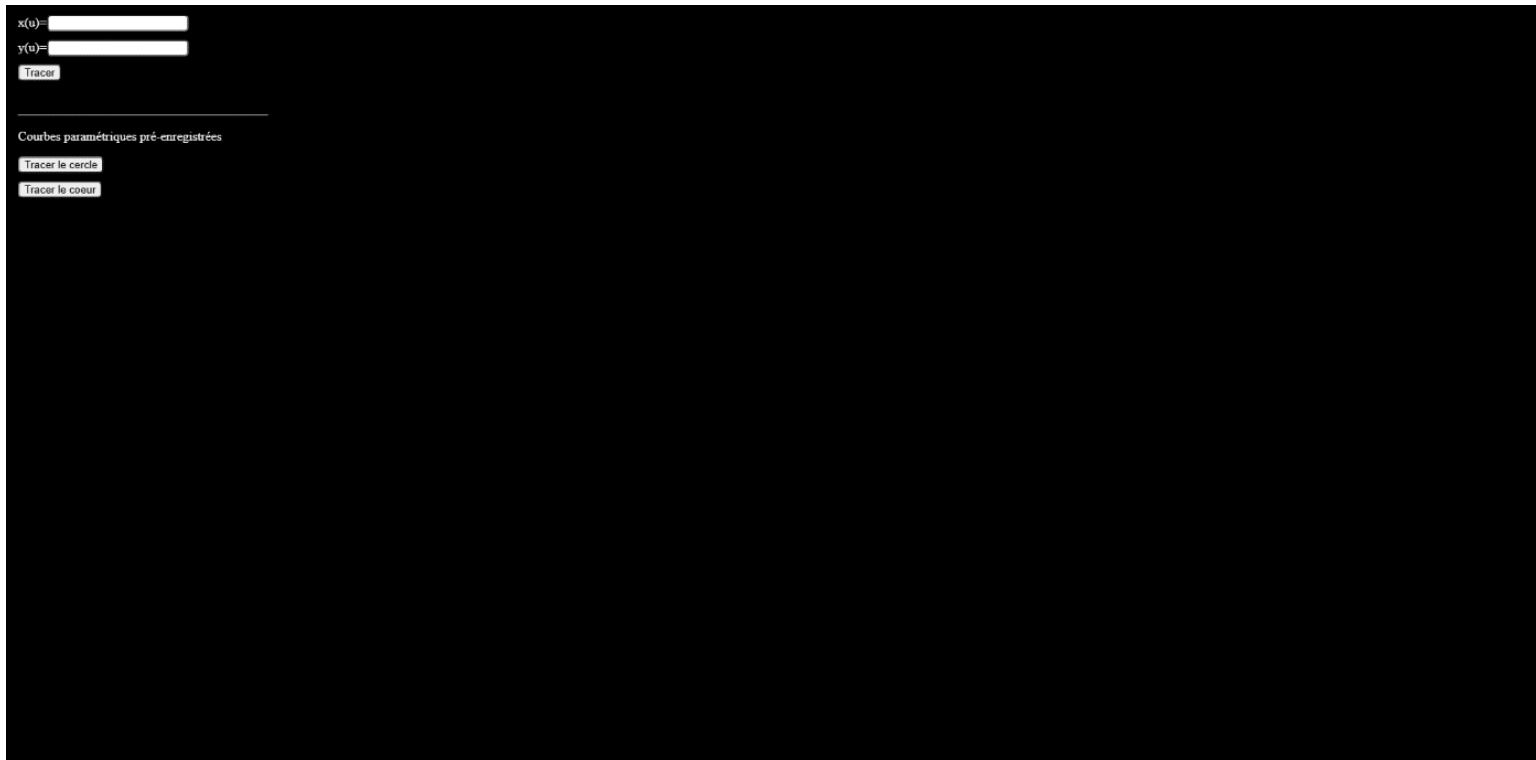
CSS :

Le code CSS utilisé est très simpliste, il se résume à un seul fichier style.css

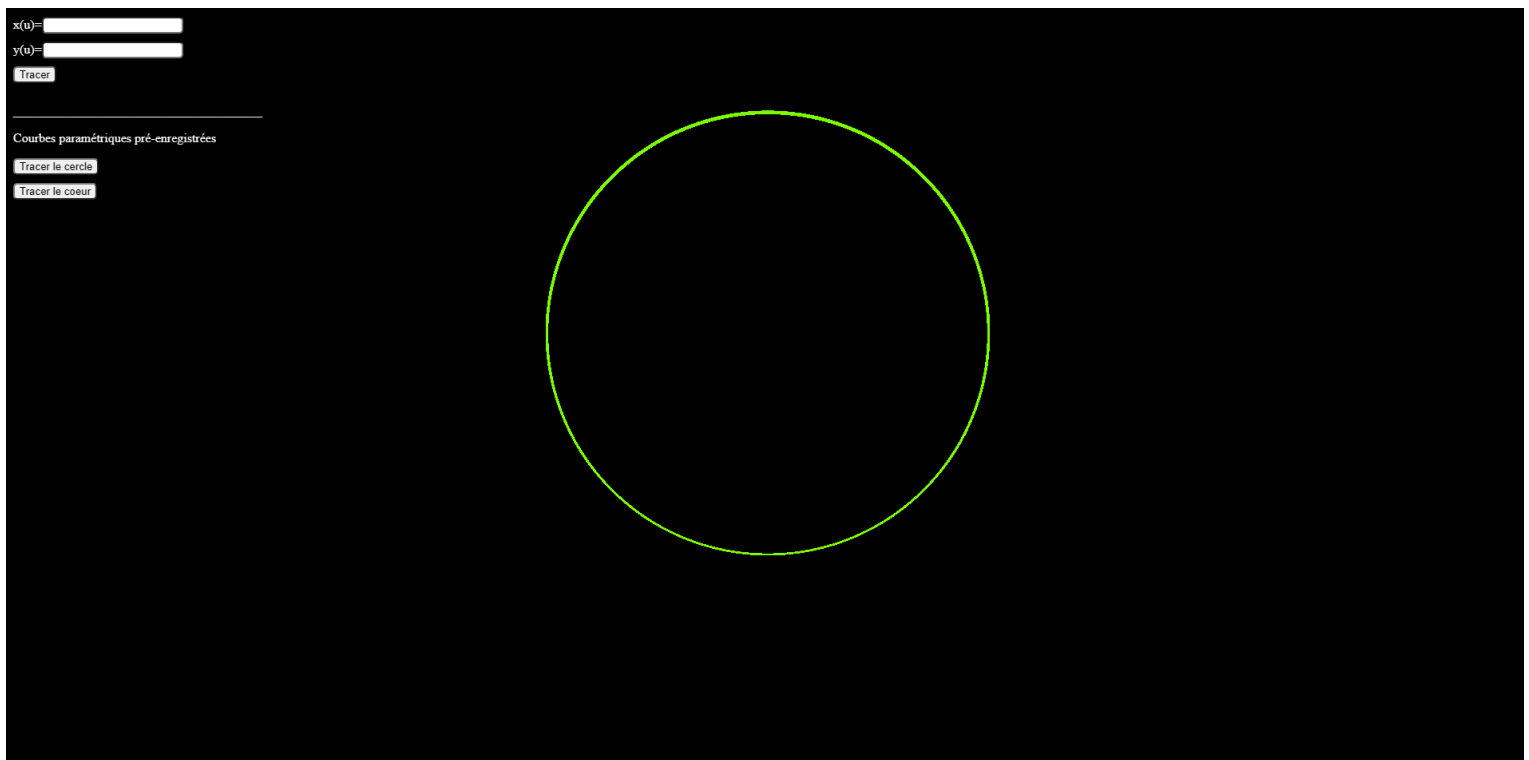
```
body {  
  margin: 0;  
}  
  
input {  
  border-radius: 5px;  
}  
  
#menu {  
  position: absolute;  
  color: white;  
  padding: 10px;  
}  
  
.options{  
  padding: 5px;  
}
```

Guide pour l'utilisateur :

En ouvrant le fichier index.html l'utilisateur se retrouve sur cette page

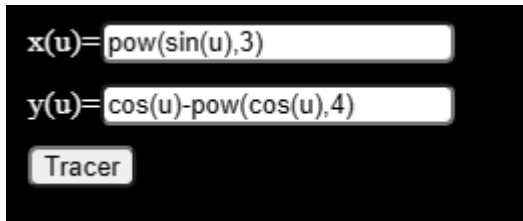


L'utilisateur a ici plusieurs choix, soit il clique sur l'une des deux courbes génériques par exemple le cercle :



Il peut également tracer la courbe qu'il souhaite en rentrant ses fonctions $x(u)$ et $y(u)$ (pas besoin de mettre Math.pow ou Math.sin, la librairie mathjs se charge de reconnaître les outils mathématiques).

Exemple d'utilisation avec la courbe paramétrique du cœur :



A screenshot of a web application interface with a black background. It features two input fields for parametric equations. The first field is labeled $x(u)=$ and contains the text `pow(sin(u),3)`. The second field is labeled $y(u)=$ and contains the text `cos(u)-pow(cos(u),4)`. Below these fields is a button labeled "Tracer".

Après avoir cliqué sur tracer il faut attendre quelques secondes car le traitement n'est pas instantané !

Voici le résultat :

