Stage DUT 2020

Visualisation progressive en 3D de données astronomiques dans un navigateur

Par : Rova RASOANAIVO
Maître de stage : André SCHAAFF
Tuteur enseignant : Pierre-Frédéric VILLARD

CENTRE DE DONNÉES ASTRONOMIQUES DE STRASBOURG

UNIVERSITÉ DE LORRAINE

IUT Saint-Dié-des-Vosges

Sommaire

Contexte du stage

État de l’application

Mise en oeuvre technique

Résultats et bilan

Perspectives du projet

Introduction

- Stage DUT Informatique à l'IUT de Saint-Dié-des-Vosges
- Visualisation 3D de données astronomiques dans le navigateur
- Recherche et développement
- Peaufiner l'application pour un potentiel déploiement

Mots-clés : Simulation, Three.js, Octree, Performances, Navigateur

Étude de l'existant

- Jasmine : ***J**avascript **AS**tronomical data **MINE**r* (2014)
- Nuages de points
  - jeux de données astronomiques fournies par le CDS
  - Coordonnées (X,Y,Z) mais pas que…
  - moteur de rendu Three.js
- Jasmine : **lire -> traiter -> afficher**
- Interaction, déplacement dans la simulation, etc.

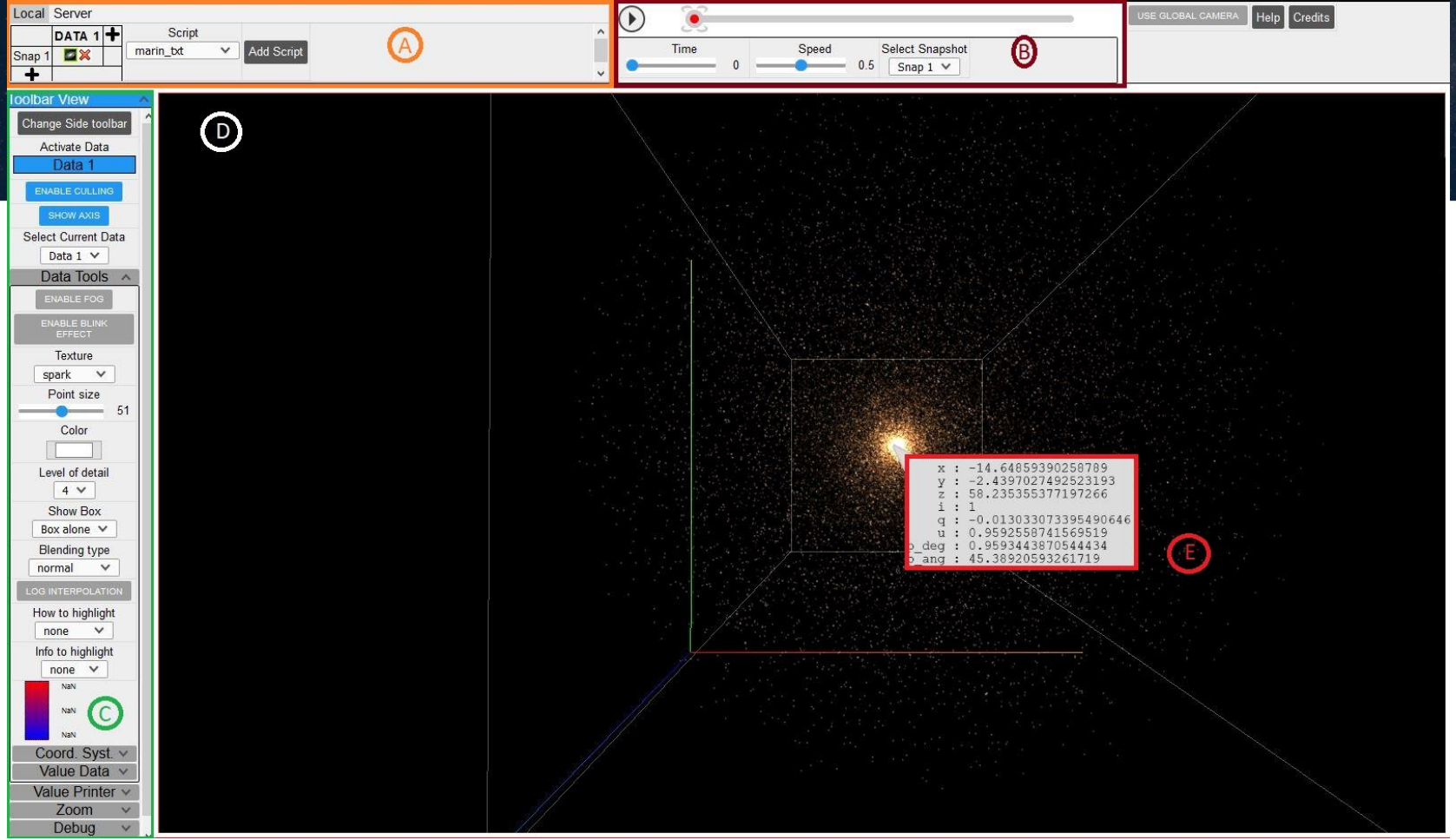
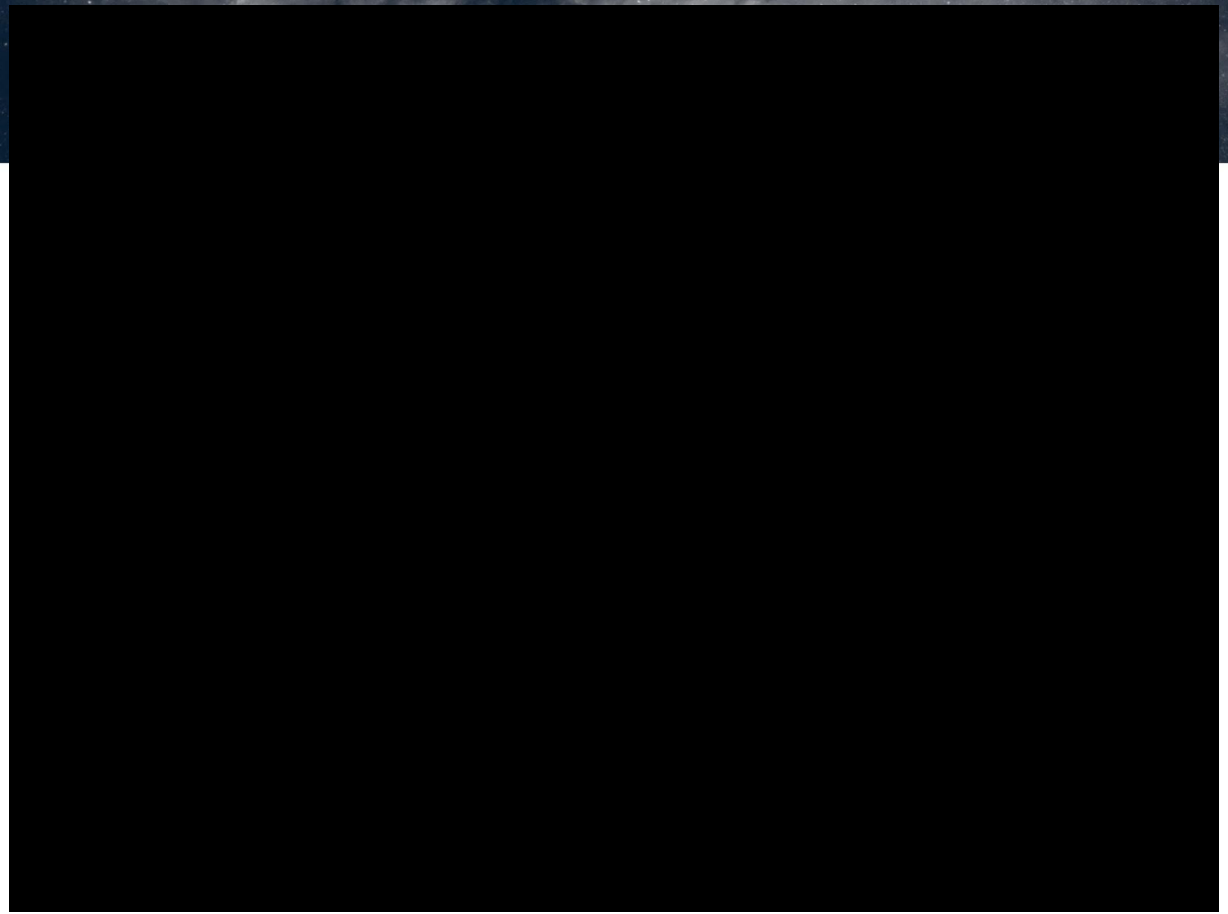


Figure : Vue de l'application côté client

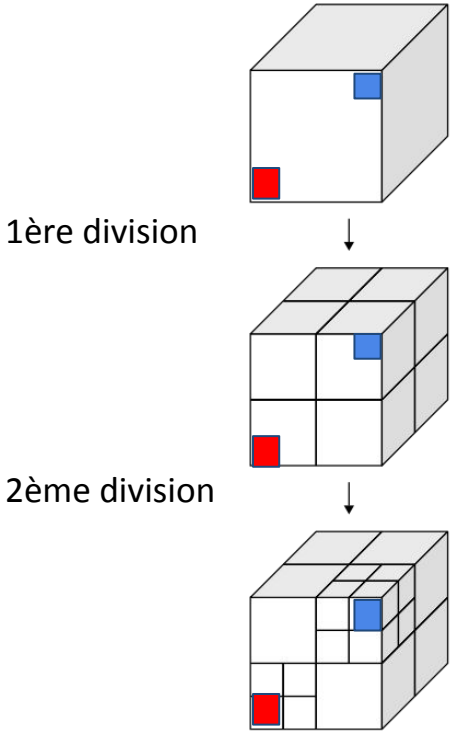


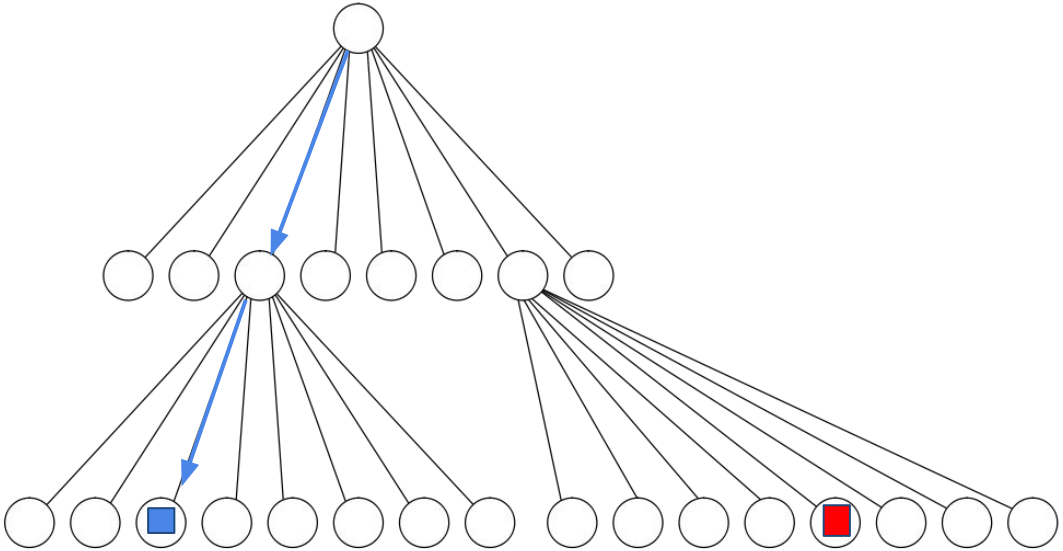
Vidéo : Utilisation de la timeline et des snapshots

Jasmine3D

- Côté client :
  - Simulation avec des fichiers en local
  - **Octree** pour les performances : lancer de rayon, occlusion,...
- Côté serveur :
  - Node.js
  - Système de fichiers en **octree**

Principe de l’octree





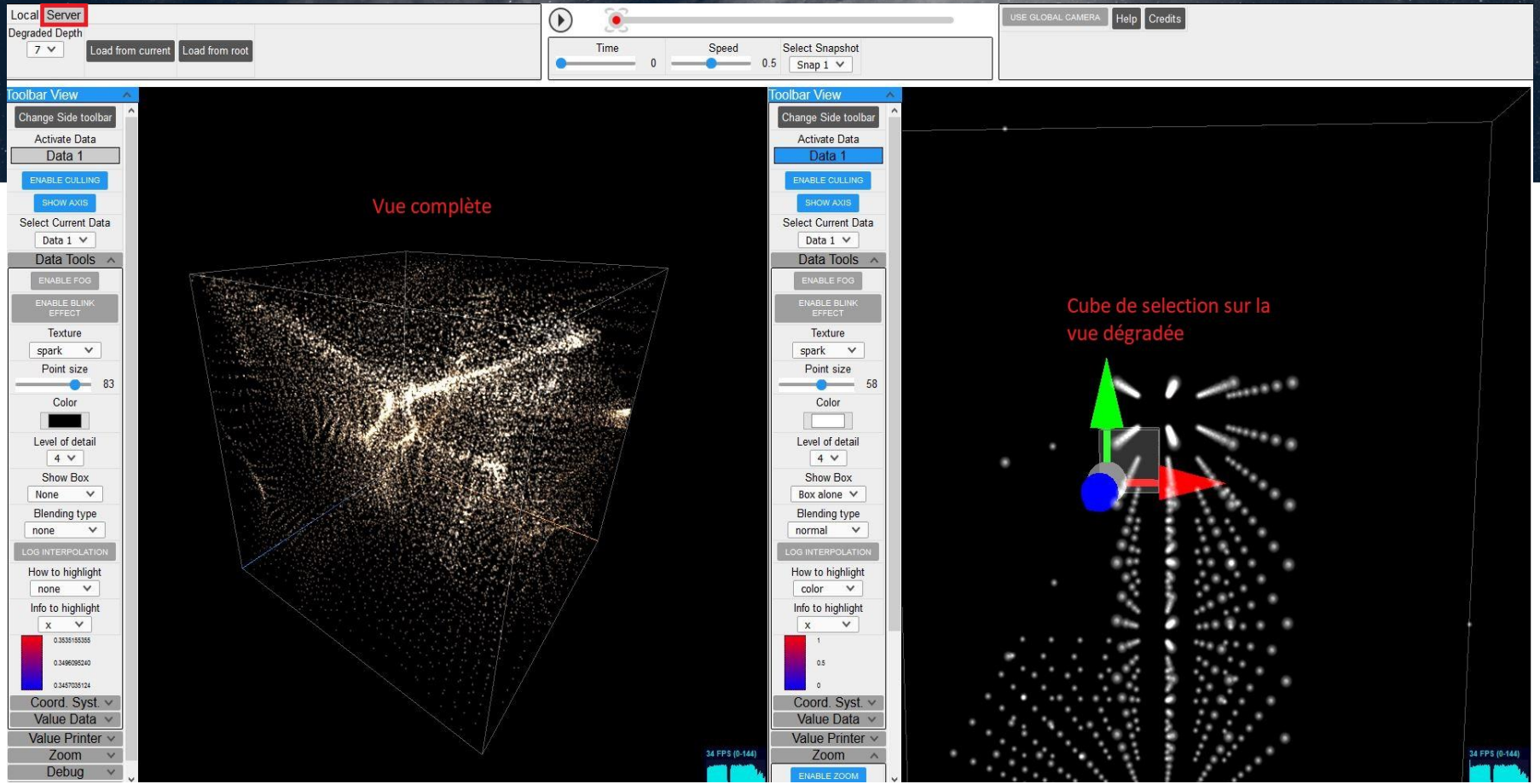


Figure : Vue de l'application - Onglet serveur

Problèmes - client

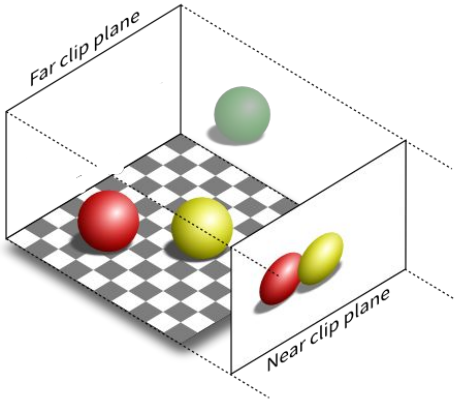
- Génération de l'Octree très lente (~2M points)

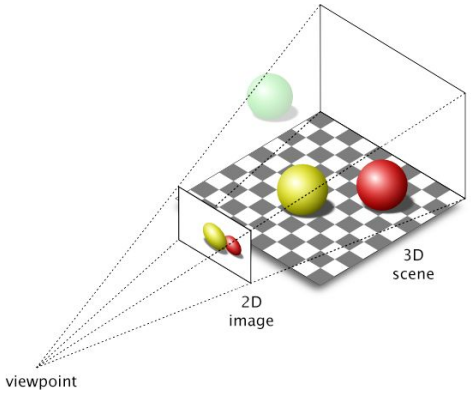
```
SPEED TEST JS : 1033 ms - chronomètre arrêté                    Octree.js:214:10
▶ Float32Array(1953125) [ 1890735, 1890736, 1890737, 1890738, 1890739,   Octree.js:215:10
1890740, 1890741, 1890742, 1890743, 1890744, … ]
```

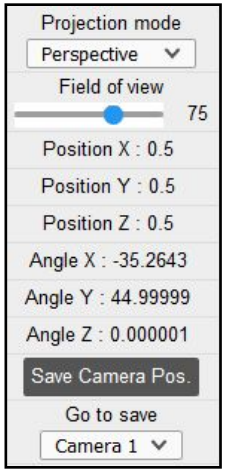
- Garbage collector JavaScript : aucun contrôle
- Personnalisation de la caméra, manque de fonctionnalités
- Vers le déploiement

Solutions techniques

- Ajout de nouvelles fonctionnalités
- Projection perspective ou orthographique







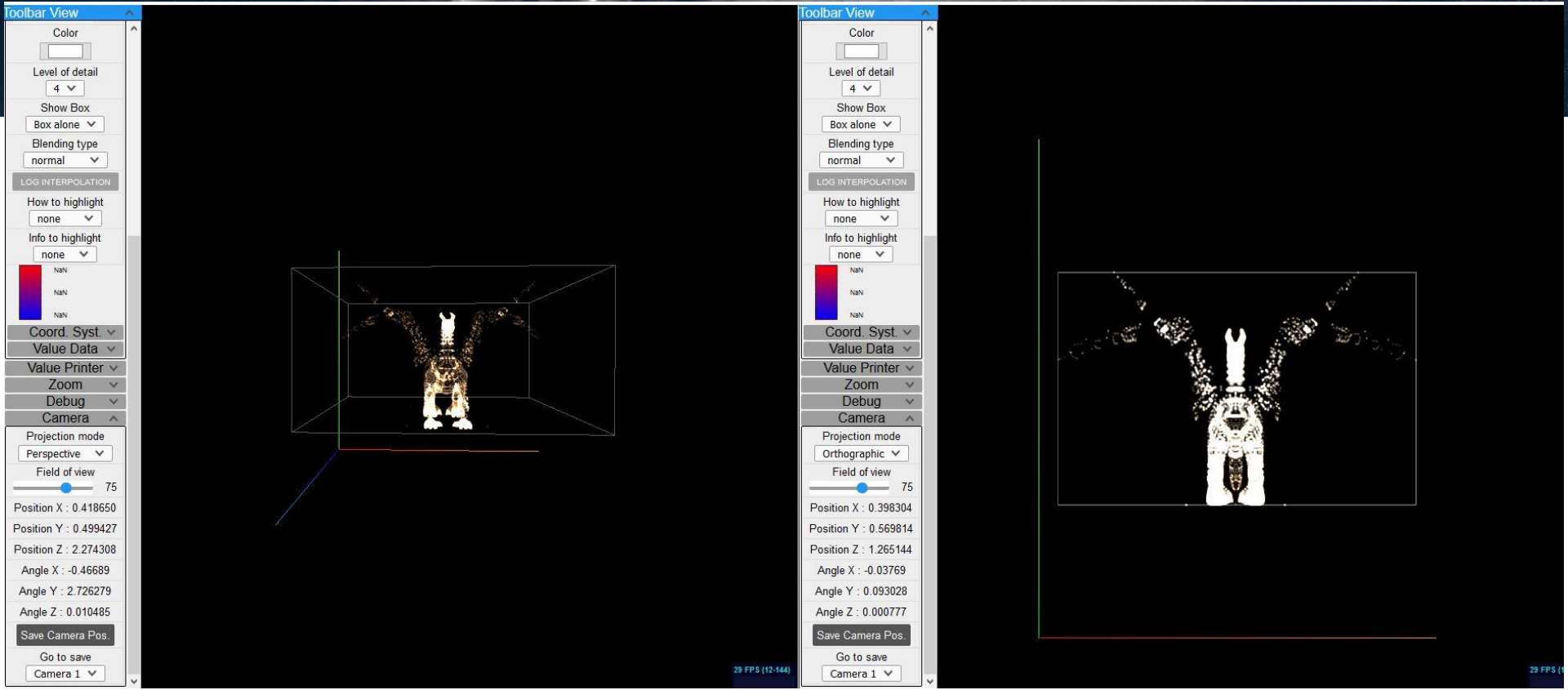
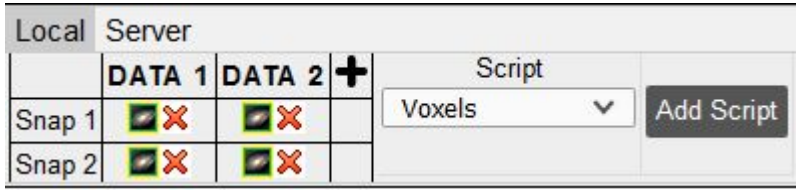


Figure : Mode multi-vue et caméra

Data Manager

- Fonctionnalités existantes mais inconnues de l'utilisateur

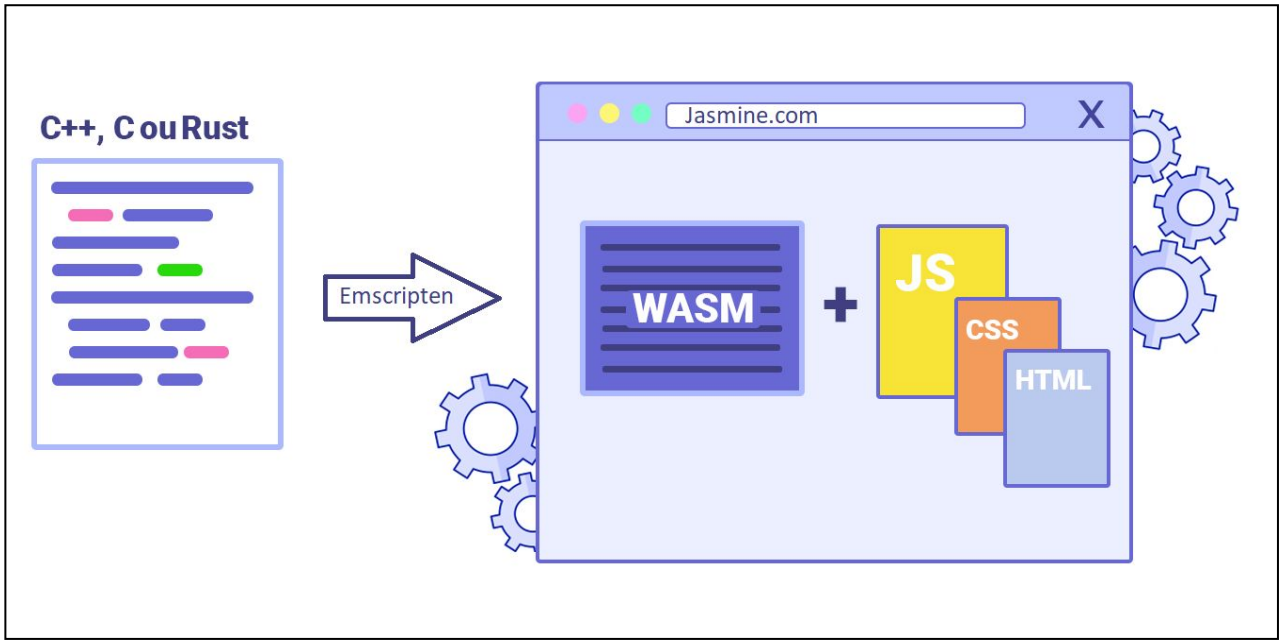


- Techniques indirectes pour "provoquer" le garbage collector :
  - déceler et supprimer les cycles de références
  - déceler les variables globales cachées

WebAssembly - Octree

- Objectifs :
  - réduire le développement JavaScript
  - performances (vitesse quasi native sur le CPU)
  - plus de contrôle mémoire
- le C++ est un bon candidat
  - Quelle chance! Le WebAssembly peut le faire “tourner” dans le navigateur
  - Comment ?

Le processus



Étapes

Pour résumer :

1. Je traduis le code en C++
2. Emscripten le compile en bytecode .wasm
3. Je l'importe dans le navigateur (via son **glue code**)
4. J'importe les fonctions C++
5. Je copie les données à traiter vers WebAssembly
6. J'appelle les fonctions importées

Problèmes - serveur

- Code à l'état de "chantier"
- Format de fichiers limité
- Mise en place du serveur trop complexe

Solutions - serveur

- Migration des scripts vers le serveur
- Refactoring
- Tests de déploiement avec ngrok

```
<!-- todo: put this in a different file!!! -->
<script>
function authenticateUser(username, password) {
  var accounts = apiService.sql(
    "SELECT * FROM users"
  );

  for (var i = 0; i < accounts.length; i++) {
    var account = accounts[i];
    if (account.username === username &&
        account.password === password)
    {
      return true;
    }
  }
  if ("true" === "true") {
    return false;
  }
}

$('#login').click(function() {
  var username = $("#username").val();
  var password = $("#password").val();

  var authenticated = authenticateUser(username, password);

  if (authenticated === true) {
    $.cookie('loggedin', 'yes', { expires: 1 });
  } else if (authenticated === false) {
    $("#error_message").show();
  }
});
</script>
```

Minification

Problèmes rencontrés

- Difficultés à comprendre le fonctionnement de l’application
- WebAssembly est relativement récent et complexe
- Peu de notions en JS et C++ -> Autoformation
- Moins de temps passé sur la partie serveur
- Adaptation au télétravail

Validation des objectifs

- [x] Réduction du développement JS
- [x] Performances
- [x] Amélioration de l’interface
- [x] Application déployable

Propositions

- Rajouter d’autres fonctionnalités WebAssembly (serveur)
- Requêtes directes vers les services du CDS (VizieR, Aladin, etc.)
- Vers le WebGPU ?

Conclusion

- Expérience très enrichissante
- Plupart des objectifs atteints
- Autoformation - liberté technique - Recherche
- Accompagnement du maître de stage André SCHAAFF
- Encore plein de potentiel...

Merci de votre attention !

Sources

Documentation officielle Emscripten :
https://emscripten.org

Documentation officielle Three.js :
https://threejs.org/docs/

Tutoriel suivi pour le WebAssembly :
https://marcoselvatici.github.io/WASM_tutorial/

WebGPU :
https://gpuweb.github.io/gpuweb/