src/src/space.js

```
/**
 * @file Describes ADSODA space
 * @author Jeff Bigot <jeff@raktres.net> after Greg Ferrar
 * @class Space
 */

import { Solid } from './solid.js'
import { Face } from './face.js'
import { Group } from './group.js'

class Space {
  /**
   * @constructor Space
   * @param {*} dim
   */
  constructor (dim, space) {
    // super("space")
    this.name = space || 'space'
    this.idx = 0
    this.dimension = dim
    this.solids = new Map()
    // this.ambientColor = false
    // this.lights = []
    this.groups = new Map()
    this.projection = []
    this.removeHidden = false
  }

  newOjectId () {
    const id = 'object' + this.idx
    this.idx += 1
    return id
  }

  /**
   * @returns JSON
   */
  exportToJSON () {
    let json = `{ "spacename" : "${this.name}" , "dimension" : ${this.dimension}
,  "solids" : [ `
    this.solids.forEach(solid => {
      json += solid.exportToJSON() + ',' // ).join(',')
    })
    json += '], "groups" : ['
    // json += [...this.groups].map(group => group.exportToJSON()).join(',')
    json += ']} '
    return json
  }

  /**
```

```
50     * create a face from a json
51     * @param {JSON}
52     * @todo add group
53     */
54    static importFromJSON (json) {
55      const space = new Space(parseInt(json.dimension), json.spacename)
56      ;[...json.solids].forEach(solid => {
57        space.suffixSolid(Solid.importFromJSON(solid))
58      })
59      if (json.groups) {
60        ;[...json.groups].forEach(agroup => {
61          const group = Group.importFromJSON(agroup, space)
62          space.suffixGroup(group)
63        })
64      }
65      // space.groups
66      // TODO: ajouter groups
67      return space
68    }
69
70    logDetail () {}
71
72    /**
73     *
74     * @param {*} solid
75     */
76    suffixGroup (grp) {
77      this.groups.set(grp.uuid, grp)
78    }
79
80    /**
81     *
82     * @param {*} solid
83     */
84    suffixSolid (solid) {
85      this.solids.set(solid.uuid, solid)
86    }
87
88    /**
89     *
90     * @param {*} light
91     */
92    // suffixLight(light) {}
93
94    /**
95     *
96     */
97    clearSolids () {
98      this.solids.clear()
99    }
100
101   /**
102    * TODO:
```

```
103     */
104   clearGroups () {
105     this.Groups.clear()
106   }
107
108   /**
109    * @todo confirmer que affectation pas utile
110    */
111   ensureSolids () {
112     this.solids.forEach(solid => {
113       solid.ensureFaces()
114       solid.ensureSilhouettes()
115     })
116   }
117   /**
118    *
119    */
120   /*
121   eliminateEmptySolids () {
122     const solids = [...this.solids].filter(solid => solid.isNonEmptySolid())
123     this.solids = solids
124   }
125   */
126
127   trigSolid (uuid) {
128     let foundInGroup = false
129     this.groups.forEach(grp => {
130       if (grp.objectList.has(uuid)) {
131         foundInGroup = true
132         console.log('found in group')
133         grp.selected = !grp.selected
134         grp.objectList.forEach(solUuid => {
135           const solid = this.solids.get(solUuid)
136           console.log('add solid', solUuid, solid )
137           solid.selected = !solid.selected
138         })
139       }
140     })
141     if (!foundInGroup) {
142       const solid = this.solids.get(uuid)
143       if (solid) {
144         console.log('add solid', uuid, solid )
145         solid.selected = !solid.selected
146       }
147     }
148   }
149
150   /**
151    *
152    * @param {*} matrix
153    * @param {*} force
154    */
155   transform (matrix, center = true, force = false) {
```

```
156     // if (this.groups.size === 0) {
157     let allSelected = true
158     let done = false
159     this.solids.forEach(solid => {
160       if (solid.selected) { allSelected = false }
161     })
162     this.groups.forEach(grp => {
163       if (grp.selected || allSelected) {
164         done = true
165         const centerptg = center ? grp.middleOf() : [0, 0, 0, 0]
166         grp.transform(matrix, centerptg, force)
167       }
168     })
169     if (!done) {
170       this.solids.forEach(solid => {
171         if (allSelected || solid.selected) {
172           const centerpt = center ? solid.middleOf() : [0, 0, 0, 0]
173           solid.transform(matrix, centerpt, force)
174         }
175       })
176     }
177   /*  } else {
178       this.groups.forEach(group => {
179         const centerptg = center ? group.middleOf() : [0, 0, 0, 0]
180         group.transform(matrix, centerptg, force)
181       })
182     }
183     */
184   }
185
186   /**
187    *
188    * @param {*} vector
189    * @param {*} force
190    */
191   translate (vector, force = false) {
192     // if (this.groups.size === 0) {
193     let allSelected = true
194     this.solids.forEach(solid => {
195       if (solid.selected) { allSelected = false }
196     })
197     this.solids.forEach(solid => {
198       if (allSelected || solid.selected) {
199         solid.translate(vector, force)
200       }
201     })
202   }
203
204   /**
205    *
206    * @param {*} axe for the moment, just the index of axe
207    * @returns return an array of solids in which hidden parts are removed
208    */
```

```
  removeHiddenSolids (axe) {
    const _tsolids = [...this.solids]
    const listOfSolids = _tsolids.map(solid => [solid.clone()])
    for (let ind = 0; ind < _tsolids.length; ind++) {
      const tempSol = _tsolids[ind]
      for (let i = 0; i < listOfSolids.length; i++) {
        if (i !== ind) {
          const tempLOS = listOfSolids[i].clone()
          const tempList = tempSol.solidsSilhouetteSubtract(tempLOS, axe)
          listOfSolids[i] = tempList
        }
      }
    }
    const flatList = listOfSolids
      .reduce((flatList, item) => flatList.concat(item), [])
      .filter(solid => solid.isNonEmptySolid())
    return flatList
  }

  /**
   * create the name of the projected space
   * @param {*} axe
   * @return text
   */
  projectName (axe) {
    return `${this.name} projection axis ${axe}`
  }

  /**
   * create the name of the projected space
   * @param {*} axe
   * @return text
   */
  axeCutName (axe) {
    return `${this.name} cut axis ${axe}`
  }

  /**
   *
   * @param {*} axe
   * @returns array of lights
   */
  // projectLights(axe) {  return [...this.lights]     }

  /**
   * project solids from space following axe
   * @param {*} axe for the moment, just the index of axe
   * @returns array of solids
   */
  projectSolids (axe) {
    // const filteredSolids = this.removeHidden
    //  ? this.removeHiddenSolids(axe)
    //  : [...this.solids]
```

```
    const projs = []
    this.solids.forEach((val, key) => projs.push(val.project(axe)))
    // if (axe === 1) console.log('projs avant filtre',projs)
    const solids = projs.reduce((solflat, item) => solflat.concat(item), [])
      .filter(solid => solid.isNonEmptySolid())
    return solids
  }

  /**
   * project solids from space following axe
   * @param {*} axe for the moment, just the index of axe
   * @returns array of solids
   */
  axeCutSolids (axe) {
    // const filteredSolids = this.removeHidden
    //  ? this.removeHiddenSolids(axe)
    //  : [...this.solids]
    const cuts = []
    this.solids.forEach((val, key) => cuts.push(val.axeCut(axe)))
    const solids = cuts.reduce((solflat, item) => solflat.concat(item), [])
      .filter(solid => solid.isNonEmptySolid())
    return solids
  }

  /**
   * Project space following axe
   * @param {*} axe for the moment, just the index of axe
   * @returns space
   */
  project (axe) {
    // if (REMOVE_HIDDEN)
    const space = new Space(this.dimension - 1)
    space.name = this.projectName(axe)
    // TODO il faut que project solids utilise filteredSolids
    const solidarray = this.projectSolids(axe)
    solidarray.forEach(solid => space.suffixSolid(solid))
    return space
  }

  /**
   * Project space following axe
   * @param {*} axe for the moment, just the index of axe
   * @returns space
   */
  axeCut (axe) {
    // if (REMOVE_HIDDEN)
    const space = new Space(this.dimension - 1)
    space.name = this.axeCutName(axe)
    // TODO il faut que project solids utilise filteredSolids
    const solidarray = this.axeCutSolids(axe)
    solidarray.forEach(solid => space.suffixSolid(solid))
    return space
  }
```

```
315
316   /**
317    * Project space following axe
318    * @param {*} hyperplane for the moment, just the index of axe
319    * @returns space
320    */
321   sliceProject (hyperplane) { }
322
323   /**
324    * @todo write
325    */
326   middleOf () {}
327
328   /**
329    * @todo write
330    */
331   deleteSelectedSolids () {}
332
333   /**
334    *
335    * @param {*} halfspaces
336    * @returns solid
337    */
338   createSolid (halfspaces) {
339     const solid = new Solid(this.dimension)
340     halfspaces.forEach(HS => solid.suffixFace(new Face(HS)))
341     solid.ensureFaces()
342     this.suffixSolid(solid)
343     return solid
344   }
345 }
346
347 export { Space }
348
```