TP 2 : araps

Le but de se TP est de réaliser un algorithme araps pour la déformation de maillage. Et d'd'inclure des interactions utilisateur.

Les donné de bases

Au début on a un bout de code qui permet d'afficher un maillage. Contenant un première élément pour sélectionner via un rectangle 2D et réalisant une déformation "non réaliste".

La déformation

Il faut appliquer une déformation rigide.

Le système linéaire

```
unsigned int equationIndex = 0;
    for( unsigned int v = 0 ; v < mesh.V.size() ; ++v ) {
        for( std::map< unsigned int , double >::const_iterator it = edgeAndVertexWeights.get_weight_of_adjacent_edges_it_begin(v) ;
             it != edgeAndVertexWeights.get_weight_of_adjacent_edges_it_end(v) ; ++it) {

            unsigned int vNeighbor = it->first;

            // WHAT TO PUT HERE ??????? How to update the entries of A ?
            arapLinearSystem.A(equationIndex, v*3) = -1;
            arapLinearSystem.A(equationIndex, vNeighbor*3) = 1;
            equationIndex++;
            arapLinearSystem.A(equationIndex, v*3+1) = -1;
            arapLinearSystem.A(equationIndex, vNeighbor*3+1) = 1;
            equationIndex++;
            arapLinearSystem.A(equationIndex, v*3+2) = -1;
            arapLinearSystem.A(equationIndex, vNeighbor*3+2) = 1;
            equationIndex++;
        }
    }
    for( unsigned int v = 0 ; v < mesh.V.size() ; ++v ) {
        if(verticesHandles[v] != -1) {

            // WHAT TO PUT HERE ??????? How to update the entries of A ?
```

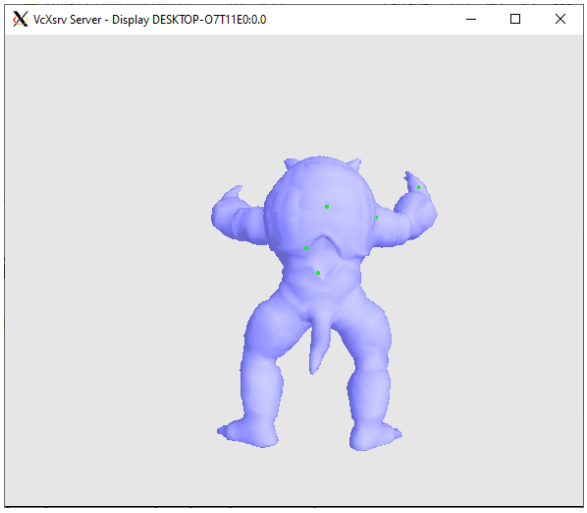
```
            arapLinearSystem.A(equationIndex, v*3) = 1;
            equationIndex++;
            arapLinearSystem.A(equationIndex, v*3+1) = 1;
            equationIndex++;
            arapLinearSystem.A(equationIndex, v*3+2) = 1;
            equationIndex++;

        }
    }

    arapLinearSystem.preprocess();
    handlesWereChanged = false;
```

L’interface utilisateur

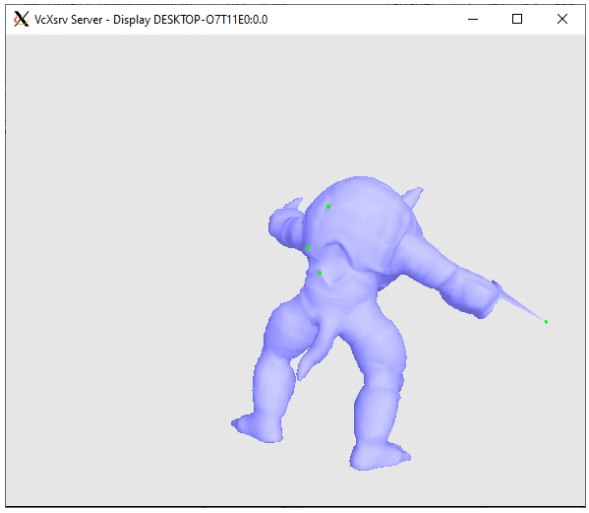
📝 **Les commandes**
clique droit : sélection par surface
clique gauche : sélection par sphère
clique milieux : sélection par point
mollette : agrandir ou diminuer le rayon (pour *surface* et *point*)
G : pour réaliser la translation

Sélection par point le plus proche

```
pair<float, int> getPlusProche(Vec3 center){
    Vec3 min = mesh.V[0];
    int indice = 0;
    for(int i = 0; i <mesh.V.size(); i++){
        if((center-min).length() > mesh.V[i].p.length()){
            min = mesh.V[i].p;
            indice = i;
        }
    }
    return pair<float, int>(0.0, indice);
}
```



Sélection par point



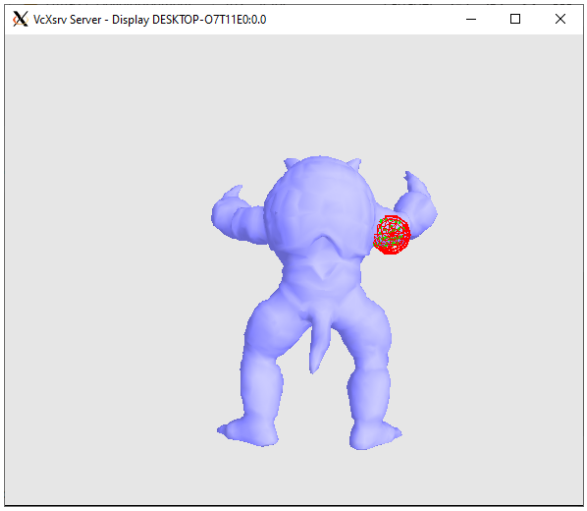
Déformation

En sélectionnant par que des points, la sélection semble moins rigides.

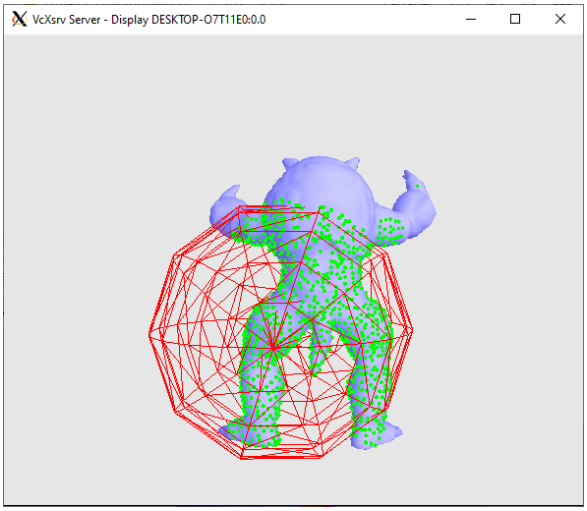
Sélection par sphère

```
void setTagForVerticesInShere() {

    GLdouble xi = pointSelect[0];
```

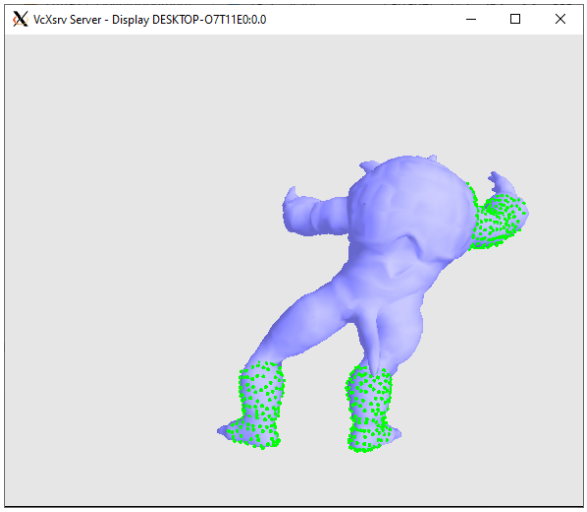
```
    GLdouble yi = pointSelect[1];
    GLdouble zi = pointSelect[2];

    for( unsigned int v = 0 ; v < mesh.V.size() ; ++v ) {
        Vec3 const & p = mesh.V[v].p;

        if(pow(p[0] - xi,2) + pow(p[1] - yi,2) + pow(p[2] - zi,2) - pow(Rayon,2) <= 0){
            verticesAreMarkedForCurrentHandle[ v ] = true;
            verticesHandles[v] = activeHandle;
            }
    }

}
```



sélection par sphère

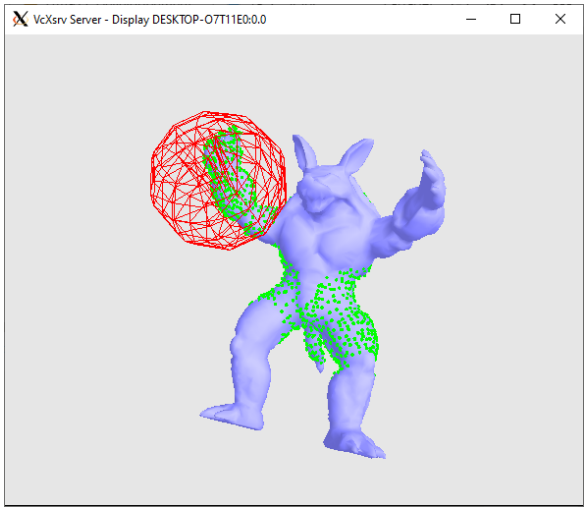


changement de taille de Rayon

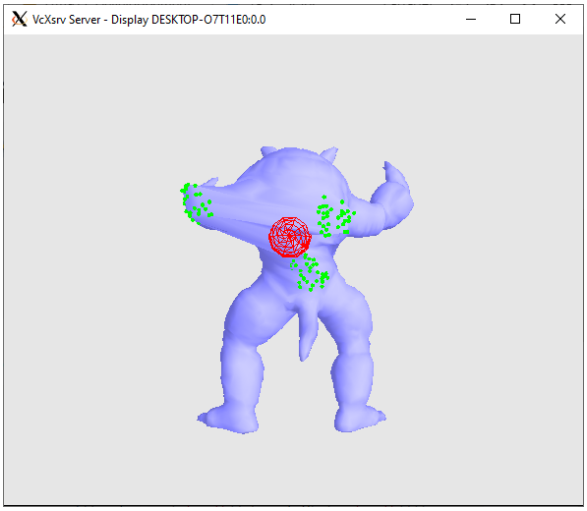


déformation

Sélection par surface



Sélection par surface



déformation

```
void collectOneRing (vector<MeshVertex> const & vertices, vector<MeshTriangle> const & triangles, vector<vector<unsigned int> > & oneR

    oneRing.clear();
    oneRing.resize(vertices.size());

    for (unsigned int i = 0; i < triangles.size(); ++i) {
        for (unsigned int j = 0; j < 3; ++j) {
            for (unsigned int k = 0; k < 3; ++k) {
                if(j != k) {
                    if(std::find(oneRing[triangles[i][j]].begin(), oneRing[triangles[i][j]].end(), triangles[i][k]) == oneRing[triangl
                        oneRing[triangles[i][j]].push_back(triangles[i][k]);
                    }
                }
            }
        }
    }
}
```

```
void getHandleSurface2( Vec3 center){
    pair<float, int> min = getPlusProche(center);
    vector<vector<unsigned int> > oneRing;
    vector<int> visite;
    visite.resize(mesh.V.size());
    fill(visite.begin(), visite.end(), 0);

    collectOneRing (mesh.V,
                    mesh.T,
                    oneRing);

    std::priority_queue<pair<float, int>> queue_vextex;
    queue_vextex.push(min);

    pair<float, int> vj = min;

    while(queue_vextex.size() > 0){
        // unsigned int i = vj.second;
        // vector<> actualoneRing = oneRing[i];
        for(auto &a : oneRing[vj.second]){

            if (visite[a] == 0)
            {
                pair<float, int> tmp;
                tmp.first = (center - mesh.V[a].p).length();
                tmp.second = a;
                queue_vextex.push(tmp);

                visite[a] = 1;
            }

        }
        vj = queue_vextex.top();
        queue_vextex.pop();
        if (vj.first <= Rayon)
        {
            verticesAreMarkedForCurrentHandle[ vj.second ] = true;
```

```
            verticesHandles[vj.second] = activeHandle;
        }
    }

}
```

Conclusion

Le code semble fonctionner, elle permet de réaliser une transformation rigide de type Araps. Cependant il manque des élément pour que cela soit complètement fonctionnelle.

- Interaction :
  - Avoir une sphère qui désélectionne les handle lorsque que le sélection diminue
  - Permettre une déformation grâce au curseur
- Algorithme :
  - Ajouter les poids
  - Retravailler pour une vrai propagation de la surfaces pour la sélection par surfaces.