Géométrie algorithmique

Les structures de données

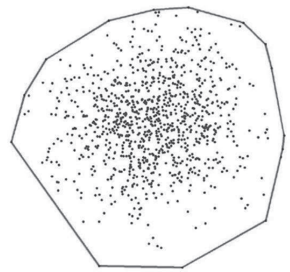
Structures de la STL:

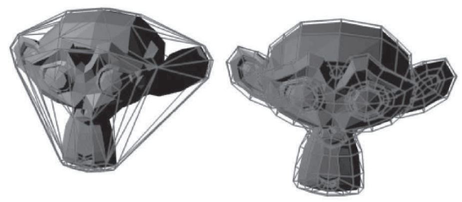
- std::vector
- std::list
- std::set
- std::map
- std::unordered_map

Ajout, Suppression, Recherche, Accès.

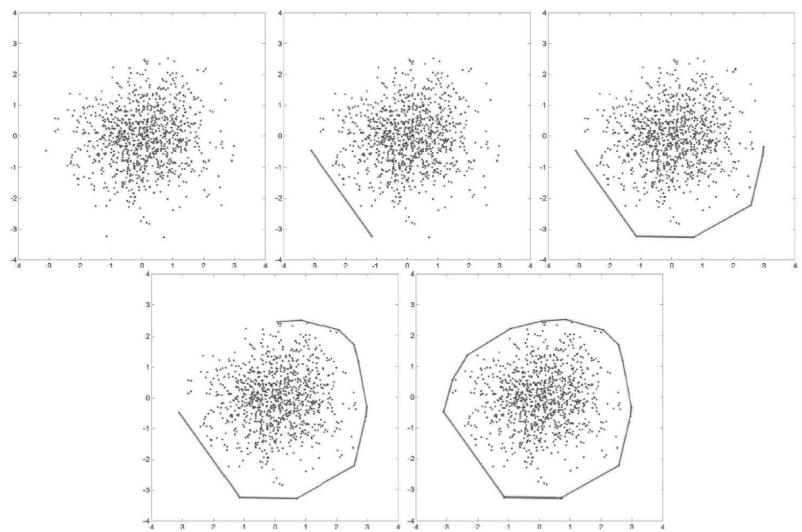
Exemple d'algorithme: Enveloppe convexe

(convex Hull)





Algorithme de Jarvis

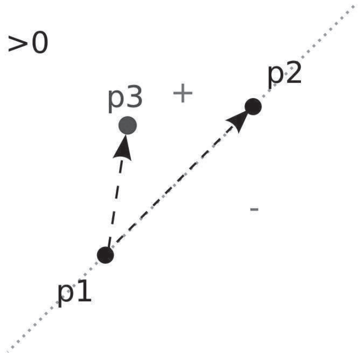


Algorithme de Jarvis

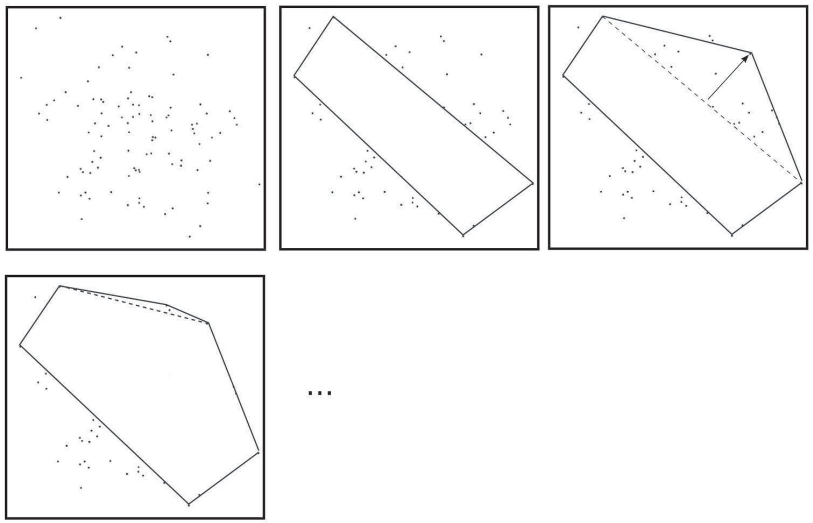
Rappel

(x3,y3) est à gauche (/droite) de [(x1,y1) , (x2,y2)] si

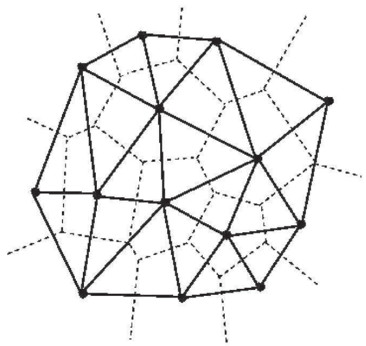
$$\begin{vmatrix} x2-x1 & x3-x1 \\ y2-y1 & y3-x1 \end{vmatrix} > 0$$

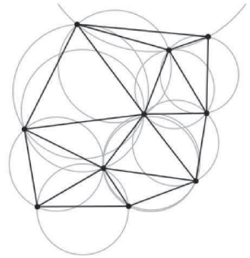


Algorithme de Quick Hull

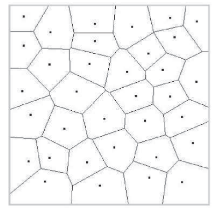


Triangulation de Delaunay



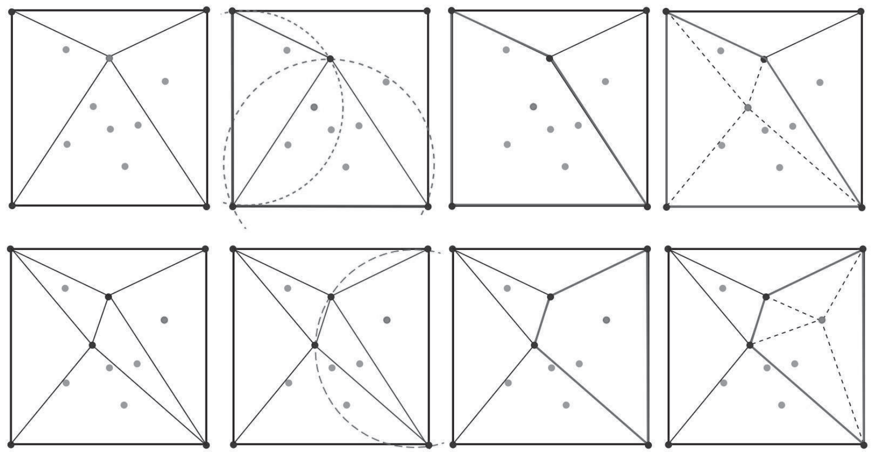


Critère de Delaunay

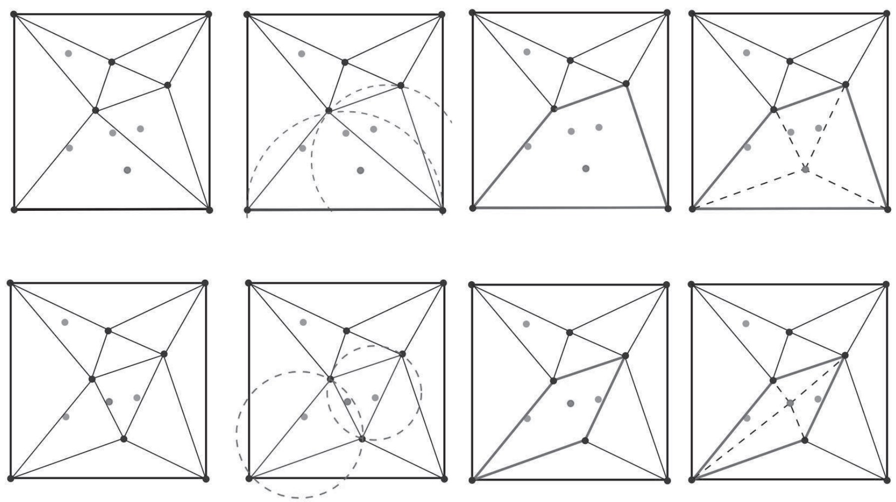


Cellules de Voronoi

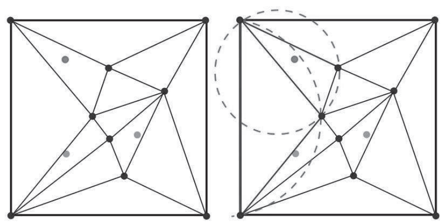
Algorithme de Bowyer Watson



Algorithme de Bowyer Watson



Algorithme de Bowyer Watson



Utilisation de CGAL

```cpp
#include <CGAL/Cartesian.h>

//contient les informations des types que l'on manipule
// "en c++: une classe de traits"
typedef CGAL::Cartesian<float> Kernel;

int main()
{
    Kernel::Vector_3 x0(1.0f,2.0f,3.0f);
    Kernel::Vector_3 x1(2.0f,1.2f,5.2f);

    std::cout<<x0<<" , "<<x1<<" , "<<x0+x1<<std::endl;

    return 0;
}
```


Utilisation de CGAL

```cpp
#include <CGAL/Cartesian.h>

//maillage 3D
#include <CGAL/Polyhedron_3.h>

typedef CGAL::Cartesian<double> Kernel;
typedef CGAL::Polyhedron_3<Kernel> Polyhedron;

int main()
{
    Kernel::Point_3 p0(0.0,0.0,0.0);
    Kernel::Point_3 p1(1.0,0.0,0.0);
    Kernel::Point_3 p2(0.0,1.0,0.0);
    Kernel::Point_3 p3(0.0,0.0,1.0);


    Polyhedron mesh;
    mesh.make_tetrahedron(p0,p1,p2,p3);


    Polyhedron::Vertex_iterator it=mesh.vertices_begin();
    Polyhedron::Vertex_iterator it_end=mesh.vertices_end();
    for(;it!=it_end;++it)
    {
        Polyhedron::Point_3 p=it->point();
        std::cout<<p<<std::endl;
    }

    return 0;
}
```

Utilisation de CGAL

```
#include <CGAL/Cartesian.h>

//maillage 3D
#include <CGAL/Polyhedron_3.h>

typedef CGAL::Cartesian<double> Kernel;
typedef CGAL::Polyhedron_3<Kernel> Polyhedron;

int main()
{
    Kernel::Point_3 p0(0.0,0.0,0.0);
    Kernel::Point_3 p1(1.0,0.0,0.0);
    Kernel::Point_3 p2(0.0,1.0,0.0);
    Kernel::Point_3 p3(0.0,0.0,1.0);


    Polyhedron mesh;
    mesh.make_tetrahedron(p0,p1,p2,p3);

    auto it=mesh.vertices_begin();
    auto it_end=mesh.vertices_end();
    for(;it!=it_end;++it)
    {
        const auto p=it->point();
        std::cout<<p<<std::endl;
    }

    return 0;
}
```


Utilisation de CGAL

```
#include <iostream>
#include <fstream>
#include <CGAL/Cartesian.h>
#include <CGAL/Polyhedron_3.h>
#include <CGAL/IO/Polyhedron_iostream.h>

typedef CGAL::Cartesian<double> Kernel;
typedef CGAL::Polyhedron_3<Kernel> Polyhedron;

int main()
{
    Polyhedron mesh;
    std::ifstream stream("../cgal_4/cube.off");
    stream>>mesh;

    std::cout<<"N_vertices = "<<mesh.size_of_vertices()<<std::endl;
    std::cout<<"N_faces = "<<mesh.size_of_facets()<<std::endl;
    std::cout<<"N_halfedges = "<<mesh.size_of_halfedges()<<std::endl;

    auto it=mesh.vertices_begin();
    auto it_end=mesh.vertices_end();
    for(;it!=it_end;++it)
    {
        const auto p=it->point();
        std::cout<<p<<std::endl;
    }

    return 0;
}
```


Utilisation de CGAL

```
typedef CGAL::Cartesian<double> Kernel;
typedef CGAL::Polyhedron_3<Kernel> Polyhedron;

int main()
{
    Polyhedron mesh;
    std::ifstream stream("../cgal_5/cube.off");
    stream>>mesh;

    Polyhedron::Halfedge_handle halfedge=mesh.halfedges_begin();

    const auto p0=halfedge->vertex()->point();
    const auto p1=halfedge->opposite()->vertex()->point();

    std::cout<<"edge 1 : ["<<p0<<","<<p1<<"]"<<std::endl;

    halfedge=halfedge->next();

    const auto p2=halfedge->vertex()->point();
    const auto p3=halfedge->opposite()->vertex()->point();

    std::cout<<"edge 2 : ["<<p2<<","<<p3<<"]"<<std::endl;


    return 0;
}
```


Utilisation de CGAL

```
typedef CGAL::Cartesian<double> Kernel;
typedef CGAL::Polyhedron_3<Kernel> Polyhedron;

int main()
{
    Polyhedron mesh;
    std::ifstream stream("../cgal_5/cube.off");
    stream>>mesh;

    auto it_face=mesh.facets_begin();
    auto it_face_end=mesh.facets_end();

    int face_number=0;
    for(;it_face!=it_face_end;++it_face)
    {
        std::cout<<"FACE : "<<face_number<<std::endl;

        auto halfedge=it_face->halfedge();
        const auto halfedge_end=halfedge;
        do
        {
            const auto p=halfedge->vertex()->point();
            std::cout<<p<<std::endl;
            halfedge=halfedge->next();

        }while(halfedge!=halfedge_end);

        face_number++;
    }

    return 0;
}
```