Universidad de Jaén

Departamento de Informática

Práctica 1

Algoritmos geométricos

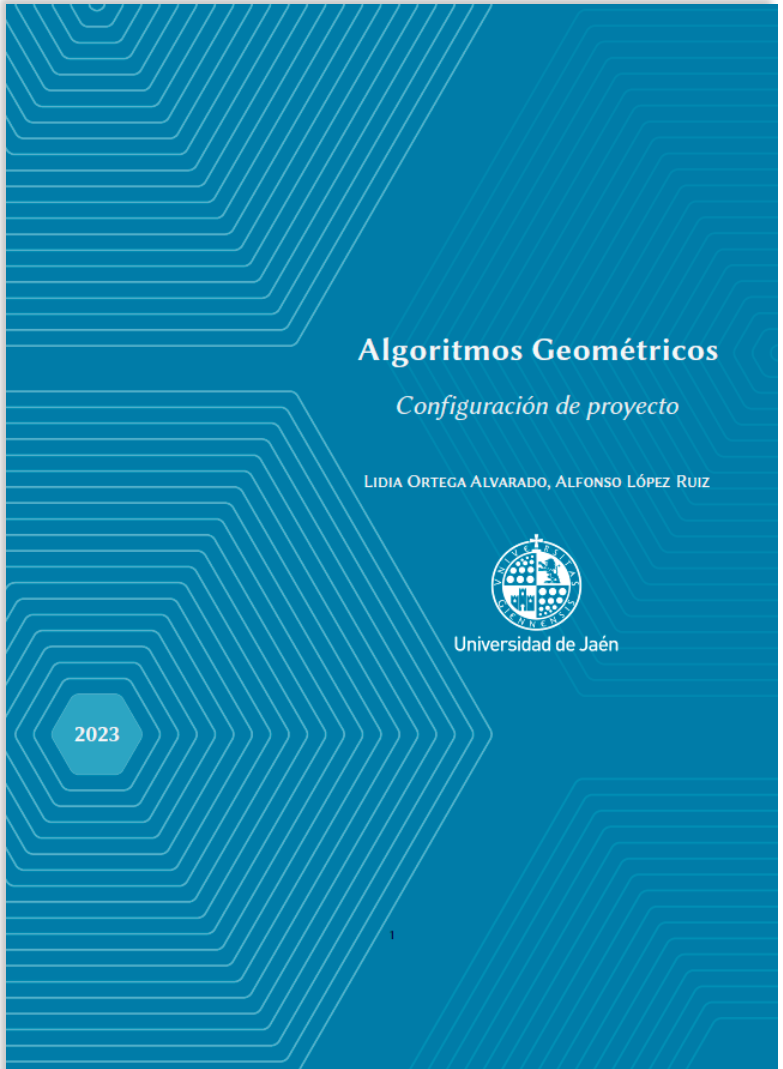
Grado en Ingeniería Informática

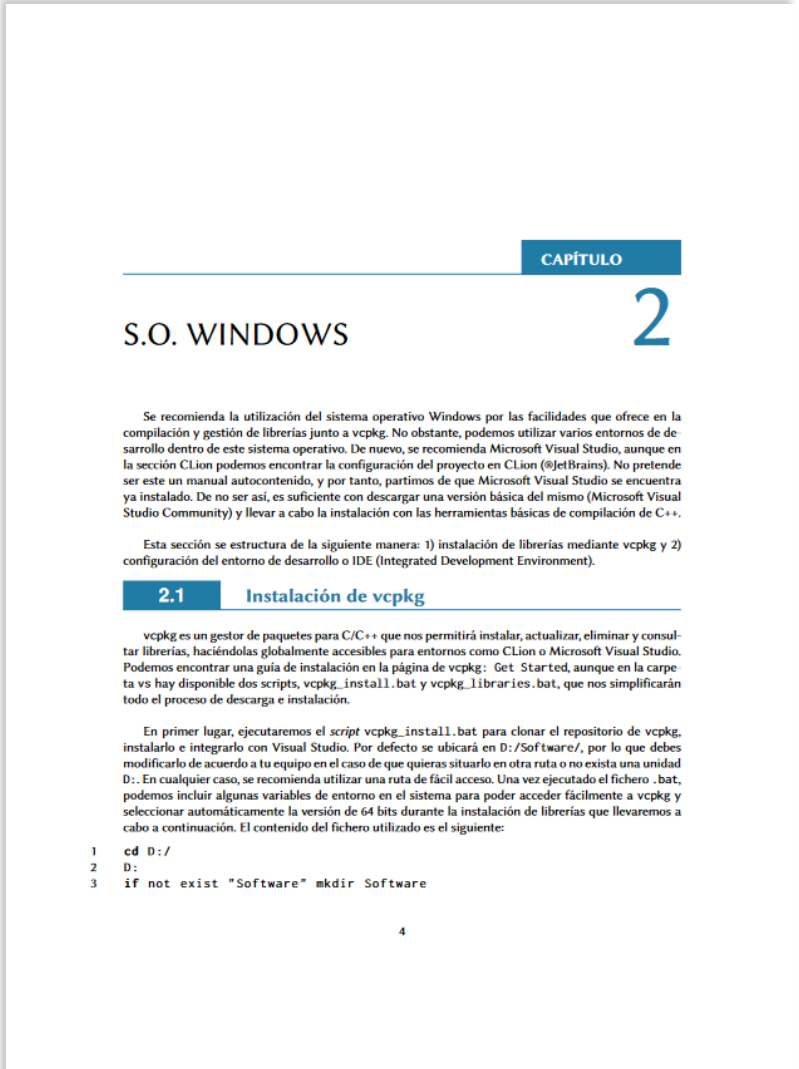
Alfonso López Ruiz

Curso 2022-2023

Installation Guide

- **Starting point:** a C++ project with OpenGL as the baseline renderer.




CAPÍTULO

2

S.O. WINDOWS

Se recomienda la utilización del sistema operativo Windows por las facilidades que ofrece en la compilación y gestión de librerías junto a vcpkg. No obstante, podemos utilizar varios entornos de desarrollo dentro de este sistema operativo. De nuevo, se recomienda Microsoft Visual Studio, aunque en la sección CLion podemos encontrar la configuración del proyecto en CLion (®JetBrains). No pretende ser este un manual autocontenido, y por tanto, partimos de que Microsoft Visual Studio se encuentra ya instalado. De no ser así, es suficiente con descargar una versión básica del mismo (Microsoft Visual Studio Community) y llevar a cabo la instalación con las herramientas básicas de compilación de C++.

Esta sección se estructura de la siguiente manera: 1) instalación de librerías mediante vcpkg y 2) configuración del entorno de desarrollo o IDE (Integrated Development Environment).

2.1 Instalación de vcpkg

vcpkg es un gestor de paquetes para C/C++ que nos permitirá instalar, actualizar, eliminar y consultar librerías, haciéndolas globalmente accesibles para entornos como CLion o Microsoft Visual Studio. Podemos encontrar una guía de instalación en la página de vcpkg: `Get Started`, aunque en la carpeta vs hay disponible dos scripts, `vcpkg_install.bat` y `vcpkg_libraries.bat`, que nos simplificarán todo el proceso de descarga e instalación.

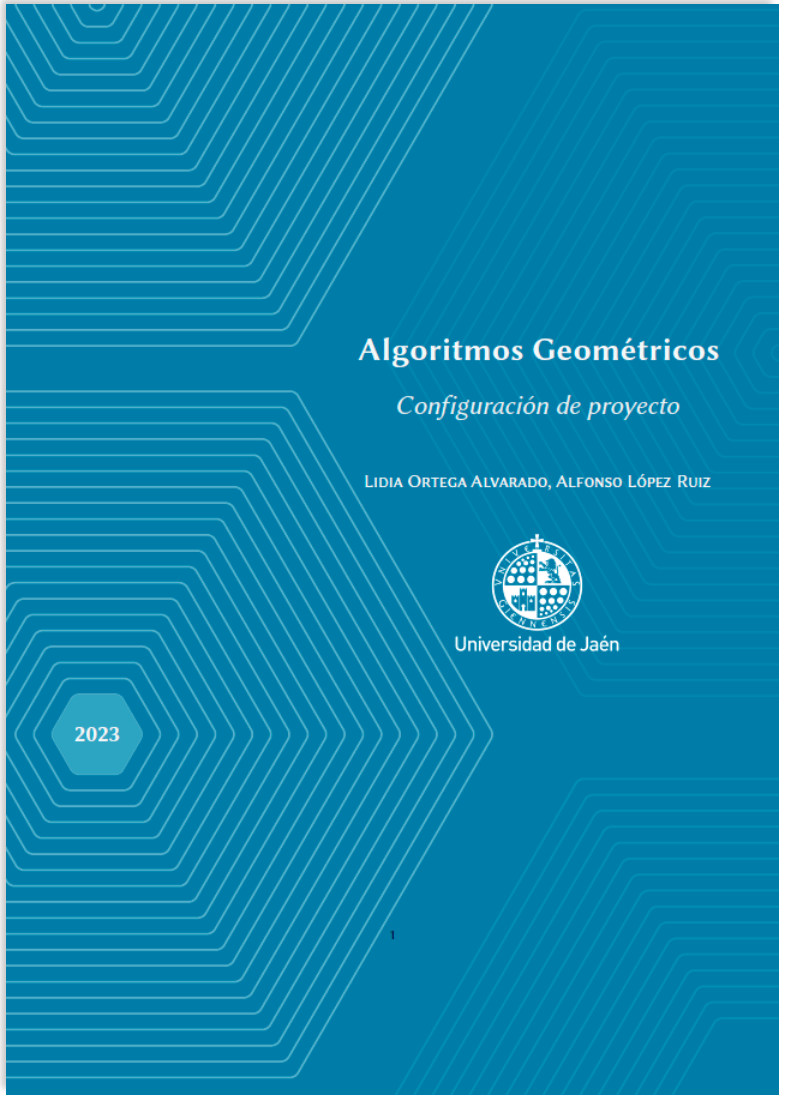
En primer lugar, ejecutaremos el *script* `vcpkg_install.bat` para clonar el repositorio de vcpkg, instalarlo e integrarlo con Visual Studio. Por defecto se ubicará en `D:/Software/`, por lo que debes modificarlo de acuerdo a tu equipo en el caso de que quieras situarlo en otra ruta o no exista una unidad `D:`. En cualquier caso, se recomienda utilizar una ruta de fácil acceso. Una vez ejecutado el fichero `.bat`, podemos incluir algunas variables de entorno en el sistema para poder acceder fácilmente a vcpkg y seleccionar automáticamente la versión de 64 bits durante la instalación de librerías que llevaremos a cabo a continuación. El contenido del fichero utilizado es el siguiente:

```
1  cd D:/
2  D:
3  if not exist "Software" mkdir Software
```

4

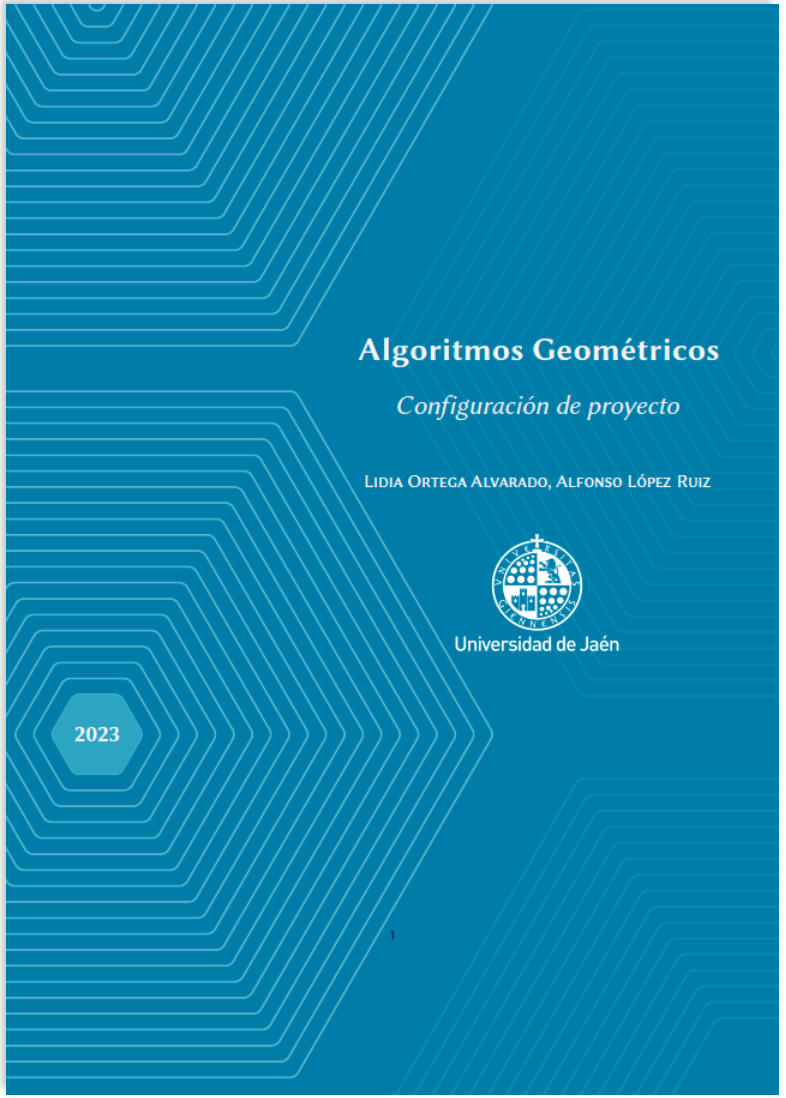
Installation Guide

- **Starting point:** a C++ project with OpenGL as the baseline renderer.
  - Read carefully the installation guide uploaded to Platea.
    - Windows + Microsoft Visual Studio + vcpkg
    - Windows + JetBrains Clion + vcpkg
    - Linux + JetBrains CLion



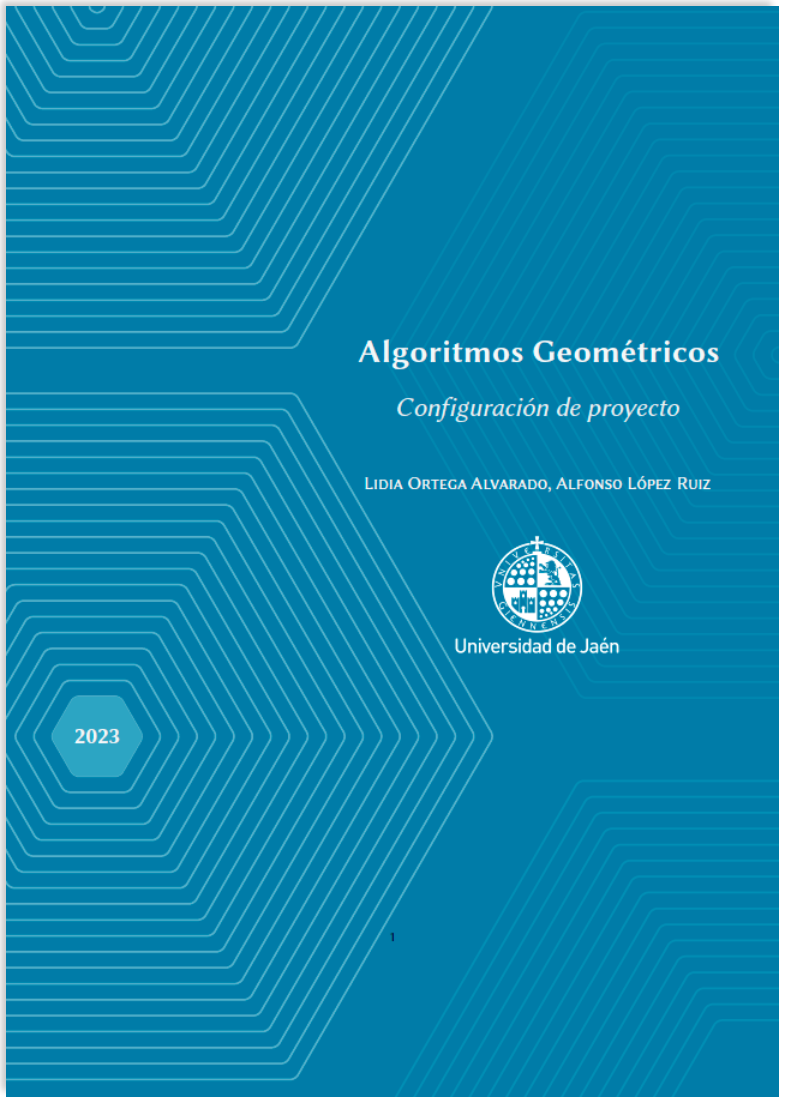
Installation Guide

- **Starting point:** a C++ project with OpenGL as the baseline renderer.
  - It is recommended to use **vcpkg** since it facilitates the installation of libraries.
  - The installation flow is already prepared in **vcpkg_install.bat**.
    - The first step is to move to a folder, D:/Software.
      - Choose one of your choice in the script, if necessary.
    - Download git repository and install it.
  - Append vcpkg to the **environment variables** (this is not included in the provided .bat files).
    - Follow the guide instructions.
  - Libraries to be installed are also prepared in **vcpkg_libraries.bat**.
  - Otherwise follow the vcpkg installation guide provided in their page.



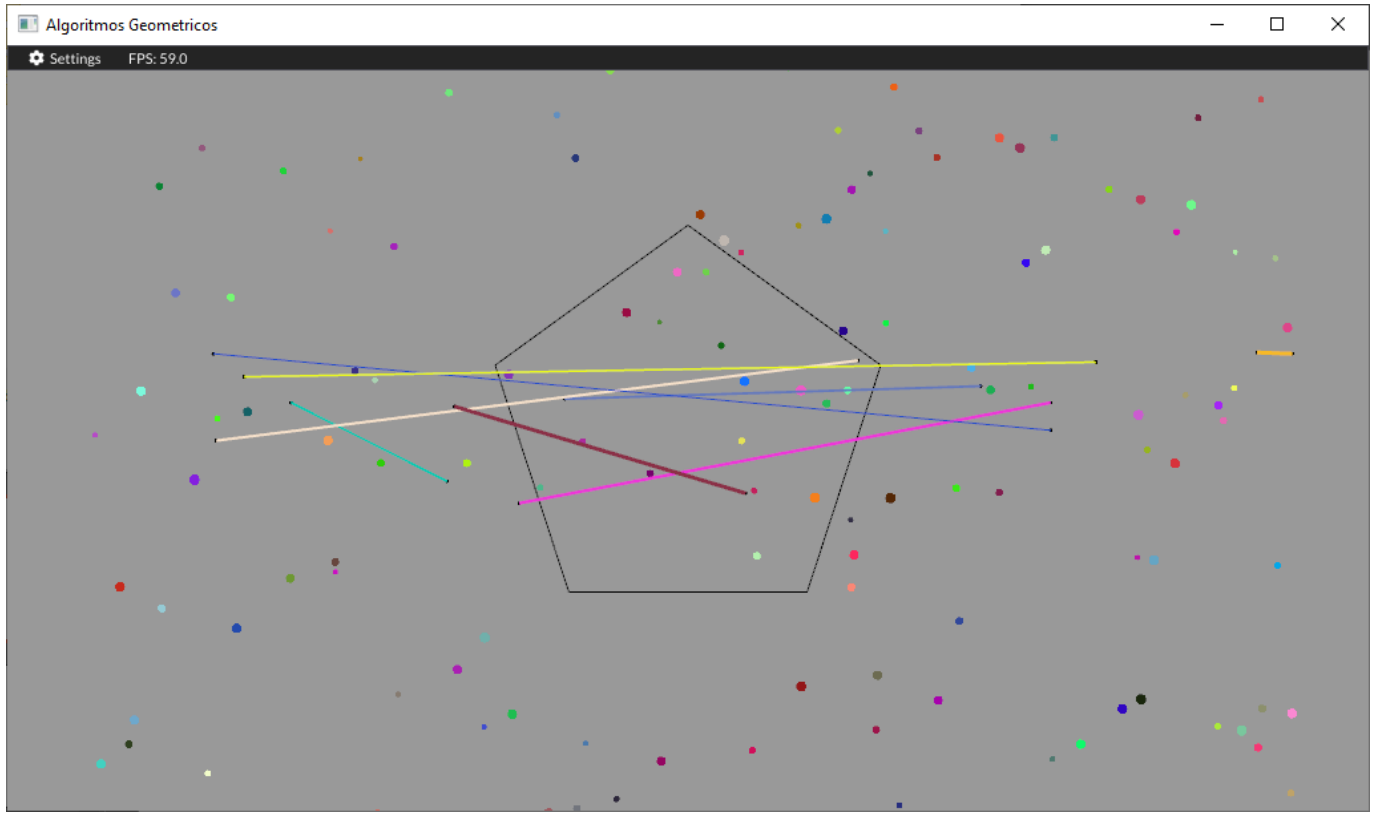
Installation Guide

- **Starting point:** a C++ project with OpenGL as the baseline renderer.
  - If Microsoft Visual Studio was opened during vcpkg installation, please close it and reopen.
  - Once ready, the project should compile with no errors.
  - The first launch should fail as there are some missing .dll files.
  - The project needs the .dll files from the installed libraries.
    - Select **x64** as the target architecture in Visual Studio.
    - The necessary .dll files have been attached to the uploaded project under a **dll folder**, following the Debug and Release structure of Visual Studio.
    - Copy both Debug and Release to x64 folder, created by Microsoft Visual Studio after the first launch.



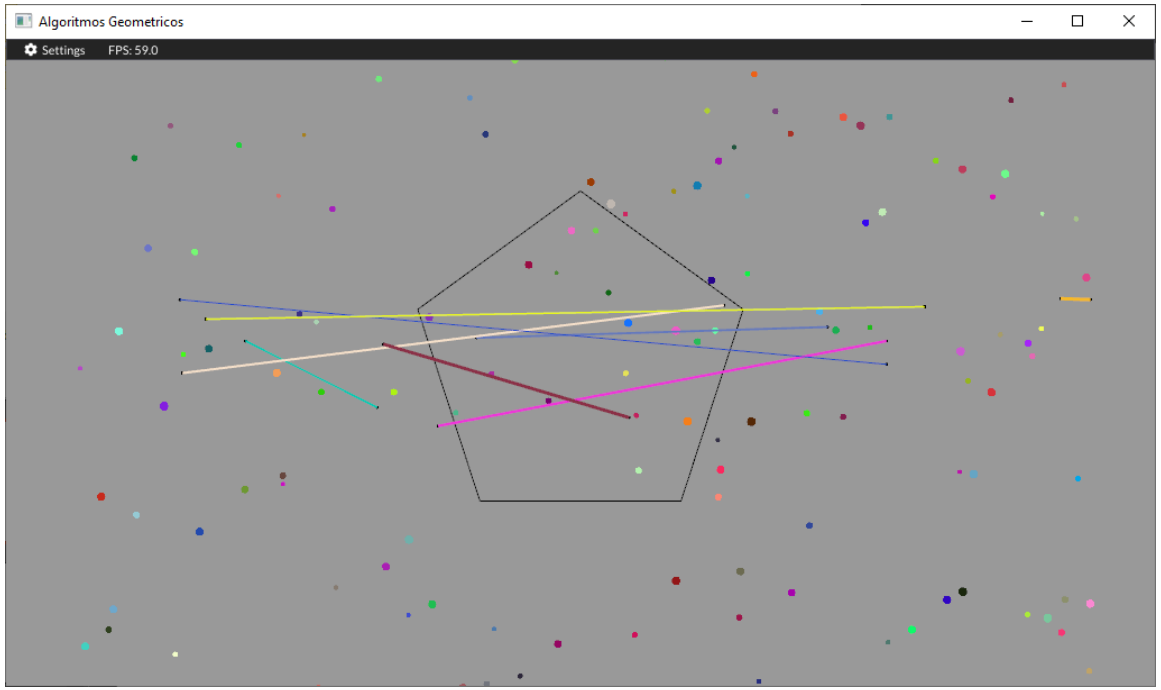
Installation Guide

- **Starting point:** a C++ project with OpenGL as the baseline renderer.
  - Everything should be up and running at this point.
  - Something like the following image should be first rendered.



Installation Guide

- **Starting point:** a C++ project with OpenGL as the baseline renderer.
  - The camera has been limited to 2D for the first practices.
    - If the interaction feels a bit weird in 2D, just change the camera initialization and pass false instead of true to enable 3D.
      - You will have to in later practices.
    - The camera interaction is described in the installation guide. It is implemented to be Unity-like (more or less).



Installation Guide

- **Starting point:** a C++ project with OpenGL as the baseline renderer.
  - Do not change code in graphics-related files.
  - It is intended to work on any mesh, primitive and shape to be implemented.
  - Files that must be modified:
    - **SceneContent**: includes the models and cameras that will be initialized by default.
      - Its only purpose is to isolate the creation of models and cameras from Renderer.
    - **Draw*:** these files are part of the objectives of this practice and the following.
    - Any file under the **Geometry** folder (or the ones that you will create, related to geometric shapes).

Practice 1

- **Objective:** to practice some of the 2D-based operations that were explained in theory lectures.
- Please, read carefully what ought to be implemented in this practice.
- Check everything is done before uploading the result.

- **Point**:
  - `Constructor based on polar coordinates.`
  - `getAlpha()`
  - `distance(p)`
  - `triangleArea(a, b)`
  - `classify(p0, p1)`
  - `slope(p)`

Practice 1

- **Objective:** to practice some of the 2D-based operations that were explained in theory lectures.
- Please, read carefully what ought to be implemented in this practice.
- Check everything is done before uploading the result.

- **Vect2D**:
  - Override `+, -, dot and scalar product` of a vector with respect to another vector.
  - Renderer of a vector.
    - It is just defined by a slope. We can draw it as a segment centred at (0, 0) with its corresponding slope.

Practice 1

- **Objective:** to practice some of the 2D-based operations that were explained in theory lectures.
- Please, read carefully what ought to be implemented in this practice.
- Check everything is done before uploading the result.

- **SegmentLine**:
  - `isHorizontal()`
  - `isVertical()`
  - `getPoint(t)`
  - `getC()`
  - `segmentIntersection(s)`
  - `impSegmentIntersection(s)`

Practice 1

- **Objective:** to practice some of the 2D-based operations that were explained in theory lectures.
- Please, read carefully what ought to be implemented in this practice.
- Check everything is done before uploading the result.

- **Line**:
  - An infinite line.
  - Trick the user perception by using very large origin and destination points.
    - E.g., (-1000, -1000, -1000) -> (1000, 1000, 1000).

Practice 1

- **Objective:** to practice some of the 2D-based operations that were explained in theory lectures.
- Please, read carefully what ought to be implemented in this practice.
- Check everything is done before uploading the result.

- **RayLine**:
  - Line clamped by origin, though it is infinite from this point.
  - Again, use large coordinates for the target point.

Practice 1

- **Objective:** to practice some of the 2D-based operations that were explained in theory lectures.
- Please, read carefully what ought to be implemented in this practice.
- Check everything is done before uploading the result.

- **Polygon**:
  - Implement constructor and storage methods with a common file format so they can interpret it properly.
  - E.g., CSV-like format, with each point in a new line, and each line composed of x, y (insert jump line).
  - Other solution which is easier to read, and a bit harder to implement: save the polygon points in a binary file.
  - `convex()`
  - `pointInConvexPolygon()`

Practice 1

- **Objective:** to practice some of the 2D-based operations that were explained in theory lectures.
- Please, read carefully what ought to be implemented in this practice.
- Check everything is done before uploading the result.

- **Vertex (from a polygon)**:
  - `convex(), concave()`
  - `next(), previous()`
  - `nextEdge(), previousEdge()`

Practice 1

- **Objective:** to practice some of the 2D-based operations that were explained in theory lectures.
- Please, read carefully what ought to be implemented in this practice.
- Check everything is done before uploading the result.

- **Point cloud**:
  - Constructors and storage methods.
    - Use the file **RandomUtilities** for randomized point cloud.
  - `centralPoint()`
  - Renderer.

Practice 1

- **Objective:** to practice some of the 2D-based operations that were explained in theory lectures.
- Please, read carefully what ought to be implemented in this practice.
- Check everything is done before uploading the result.

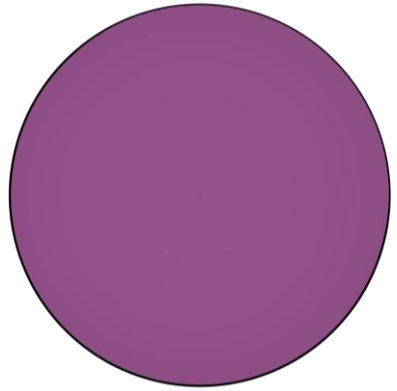
- **Circle**:
  - `isInside(p)`

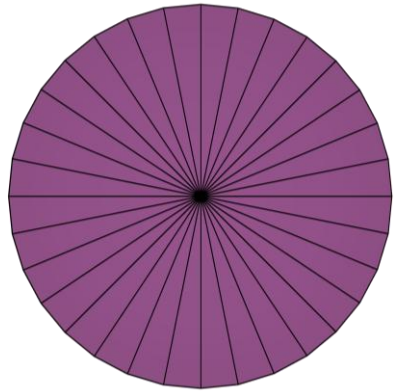
Practice 1

- **Final scenario:** you will have to compose a final scenario in **SceneContent**, consisting of previously implemented primitives.

1. **Random point cloud (100 points) + storage.**
   - Place it in a visible area within the viewport.
2. **Three random segments using the cloud points.**
3. **Select the points with minimum and maximum X and Y coordinates and create a polygon defined in counterclockwise direction.**
   - Check if its convex.
4. **Draw 3 circumferences whose centre is the cloud point which is the closest to (0, 0). Pick a random point for each circumference and make it pass over the selected point (use it to calculate the radius).**

Preferred format for rendering a circumference.





Easier, uglier, but it is also ok.

Results

- **What should be uploaded?**
  - **The whole project. Reduce the size with Compile > Clean solution.**
  - **The executable file located either under x64/Debug or x64/Release.**
    - Check it works before uploading. The Assets folder should be copied under x64/Debug or x64/Release so that the program can find these files.
- **Deadline**: February 13th, 23:59

How to implement a new rendering class

- **You can find detailed instructions in the final chapter of the uploded guide.**

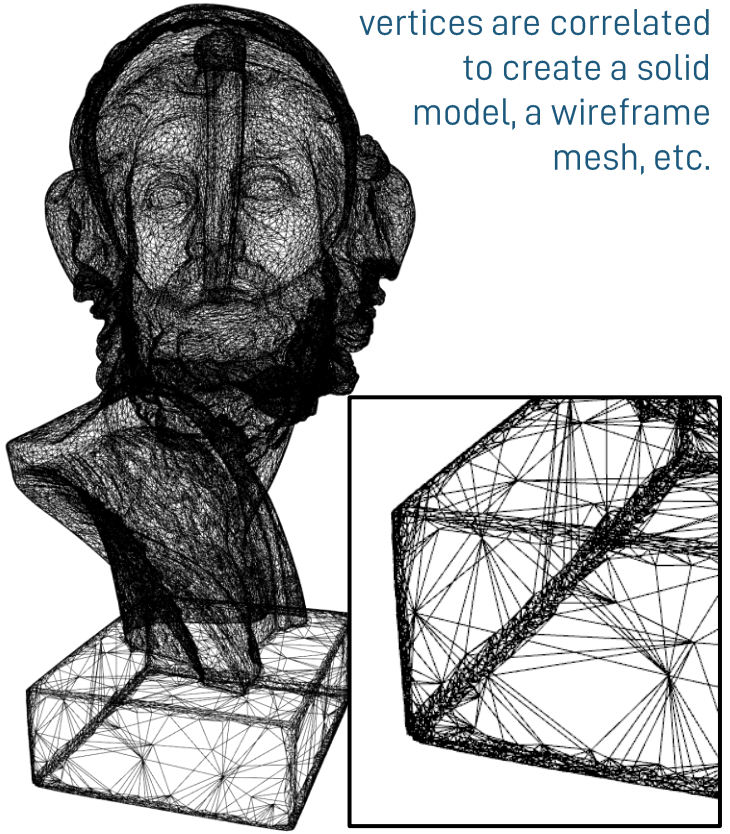
1. **Everything should be defined within the class constructor:** geometry and topology.
2. **Revise the concepts of geometry and topology.**
3. **Every model can be composed of several components.**
    - For instance, a point cloud could have two components: the point cloud itself and the convex hull calculated from it.
4. **Two vectors for each component:** geometry and topology.

How to implement a new rendering class

- **Geometry and topology.**
  - Topology is called ‘primitives’ in your installation guide to avoid overlapping with later concepts.

Geometry: raw vertices.

Topology: defines how vertices are correlated to create a solid model, a wireframe mesh, etc.



Universidad de Jaén

Departamento de Informática

Práctica 4.B

Algoritmos geométricos

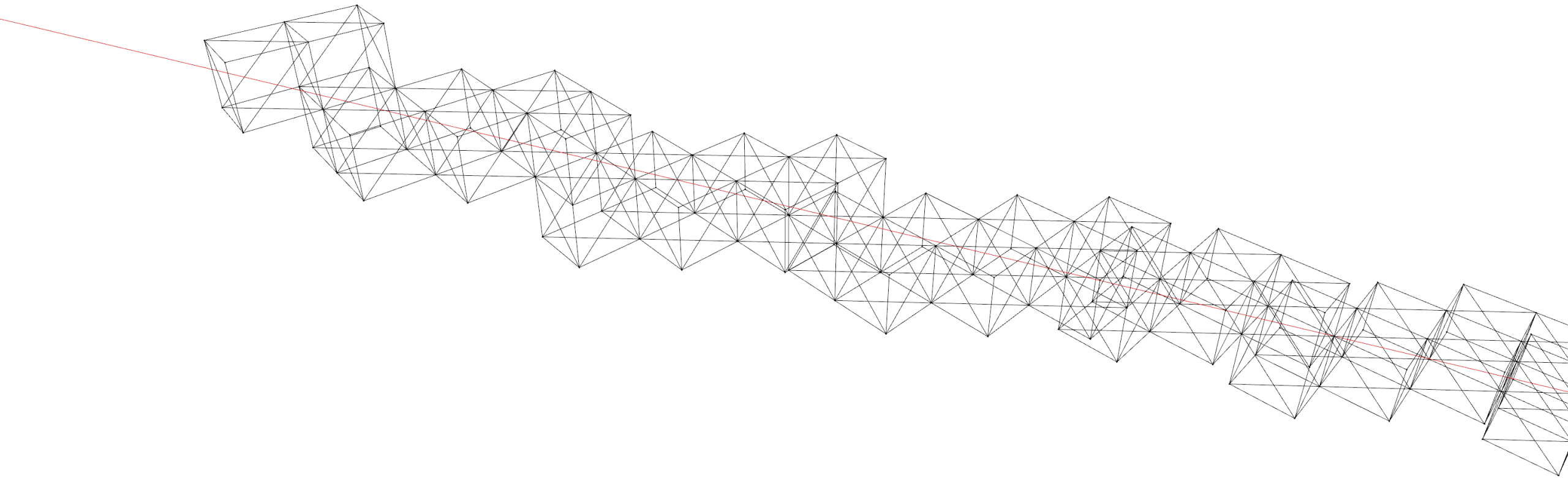
Grado en Ingeniería Informática

Alfonso López Ruiz

Curso 2022-2023

A fast voxel traversal algorithm for ray tracing

- **Lanzamos un rayo y obtenemos qué vóxeles atraviesa.**
  - Obtener dicho conjunto de vóxeles y renderizarlo:
    - `new DrawVoxelization(voxels.data(), voxels.size(), voxels[0].extent() * 2.0f)`



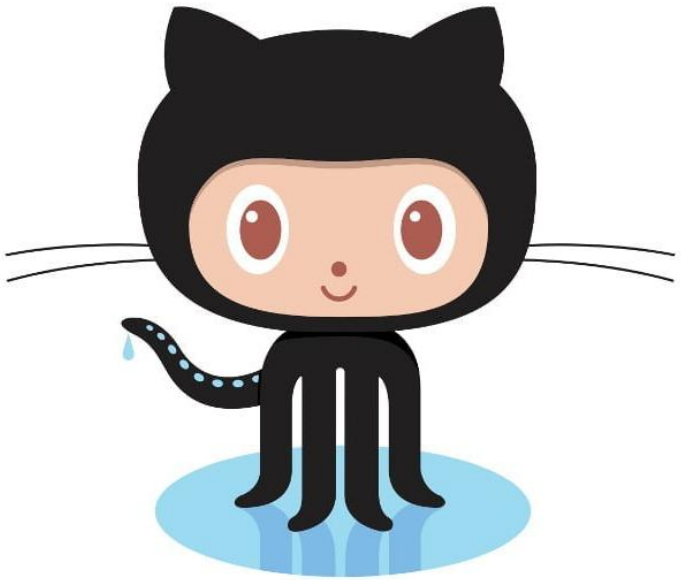
A fast voxel traversal algorithm for ray tracing

- **Lanzamos un rayo y obtenemos qué vóxeles atraviesa**.
  - Obtener dicho conjunto de vóxeles y renderizarlo:
    - `new DrawVoxelization(voxels.data(), voxels.size(), voxels[0].extent() * 2.0f)`
  - Los vóxeles grises pueden almacenar un conjunto de triángulos (opcional).
    - Ejecutar test ray-triángulo en dichas primitivas.
    - Determinar qué primitivas son intersectadas y renderizarlas.
      - **DrawTriangle**

A fast voxel traversal algorithm for ray tracing

Algoritmo:

cgyurgyik/**fast-voxel-traversal-algorithm**



```
108     size_t current_Z_index = MAX(1, std::ceil(ray_start.z() - grid.minBound().z() / grid.voxelSizeZ()));
109     const size_t end_Z_index = MAX(1, std::ceil(ray_end.z() - grid.minBound().z() / grid.voxelSizeZ()));
110     int stepZ;
111     value_type tDeltaZ;
112     value_type tMaxZ;
113     if (ray.direction().z() > 0.0) {
114         stepZ = 1;
115         tDeltaZ = grid.voxelSizeZ() / ray.direction().z();
116         tMaxZ = tMin + (grid.minBound().z() + current_Z_index * grid.voxelSizeZ()
117                         - ray_start.z()) / ray.direction().z();
118     } else if (ray.direction().z() < 0.0) {
119         stepZ = -1;
120         tDeltaZ = grid.voxelSizeZ() / -ray.direction().z();
121         const size_t previous_Z_index = current_Z_index - 1;
122         tMaxZ = tMin + (grid.minBound().z() + previous_Z_index * grid.voxelSizeZ()
123                         - ray_start.z()) / ray.direction().z();
124     } else {
125         stepZ = 0;
126         tDeltaZ = tMax;
127         tMaxZ = tMax;
128     }
129
130     while (current_X_index != end_X_index || current_Y_index != end_Y_index || current_Z_index != end_Z_index) {
131         if (tMaxX < tMaxY && tMaxX < tMaxZ) {
132             // X-axis traversal.
133             current_X_index += stepX;
134             tMaxX += tDeltaX;
135         } else if (tMaxY < tMaxZ) {
136             // Y-axis traversal.
137             current_Y_index += stepY;
138             tMaxY += tDeltaY;
139         } else {
140             // Z-axis traversal.
141             current_Z_index += stepZ;
142             tMaxZ += tDeltaZ;
143         }
144     }
145 }
```

A fast voxel traversal algorithm for ray tracing

- **Algoritmo:**

1. Comprobar que el punto está dentro de la voxelización.
   - `glm::all(glm::greaterThanEqual(point, 0)) && …`

A fast voxel traversal algorithm for ray tracing

- **Algoritmo:**

2

- Obtener avance en vóxel.
  - Si el rayo tiene una coordenada x positiva, siempre avanzaremos +1 en X cuando sea necesario.
  - Si el rayo tiene una coordenada x negativa, siempre avanzaremos -1 en X cuando sea necesario.
  - `glm::sign(rayDirection)`

A fast voxel traversal algorithm for ray tracing

- **Algoritmo:**

3

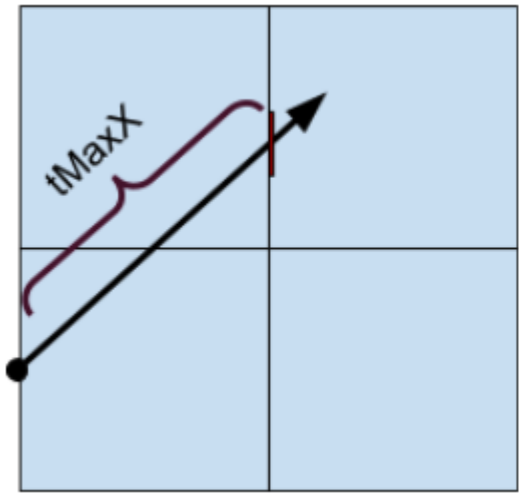
- Obtener la longitud del avance, *tDelta*.
  - *tDelta* dependerá del tamaño de vóxel y de la dirección del rayo.
  - Nótese que se trata de un valor positivo siempre (es un *t* a lo largo del rayo).

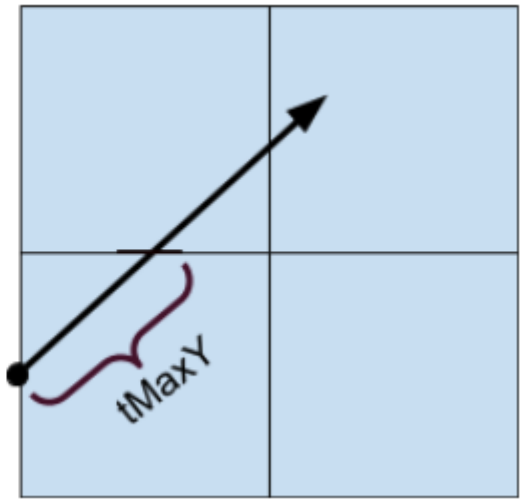
A fast voxel traversal algorithm for ray tracing

- **Algoritmo:**

4

- Obtener el t inicial, $tMax$.
  - La situación más normal es que el origen del rayo no se encuentre en las fronteras de un vóxel, sino dentro de él.
  - Sin embargo, los incrementos antes calculados se aplican desde los límites del vóxel.



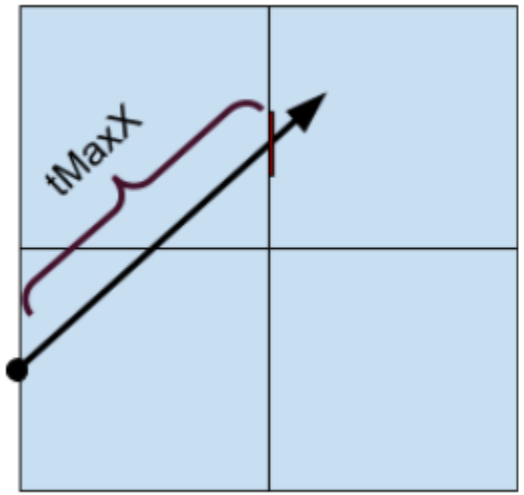


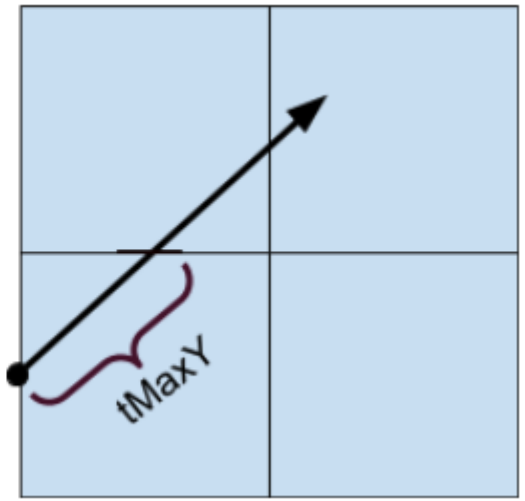
A fast voxel traversal algorithm for ray tracing

- **Algoritmo:**

4

- Obtener el *t* inicial, *tMax*.
  - El rayo comienza en el vóxel *i, j, k*.
  - Es posible calcular el mínimo punto del vóxel *i, j, k*: *voxelization.min + voxelSize * indices*
  - ¿Cuál es la *t*? `(Punto obtenido – Origen del rayo) / Dirección del rayo`



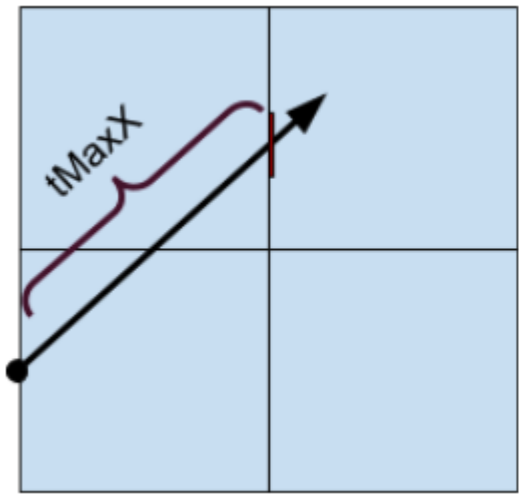


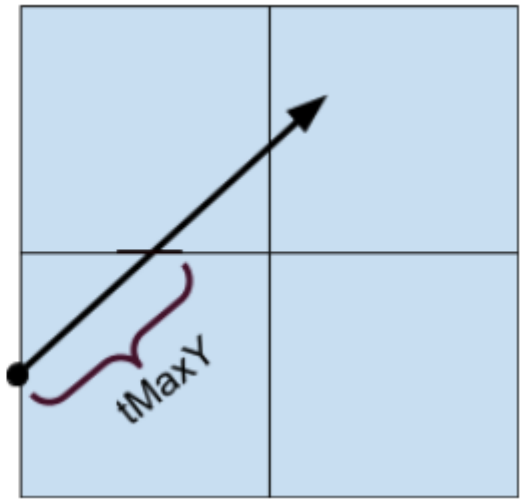
A fast voxel traversal algorithm for ray tracing

- **Algoritmo:**

4

- Obtener el t inicial, $tMax$.
  - Importante: si la dirección del rayo es negativa, se utiliza $i - 1, j - 1, k - 1$.



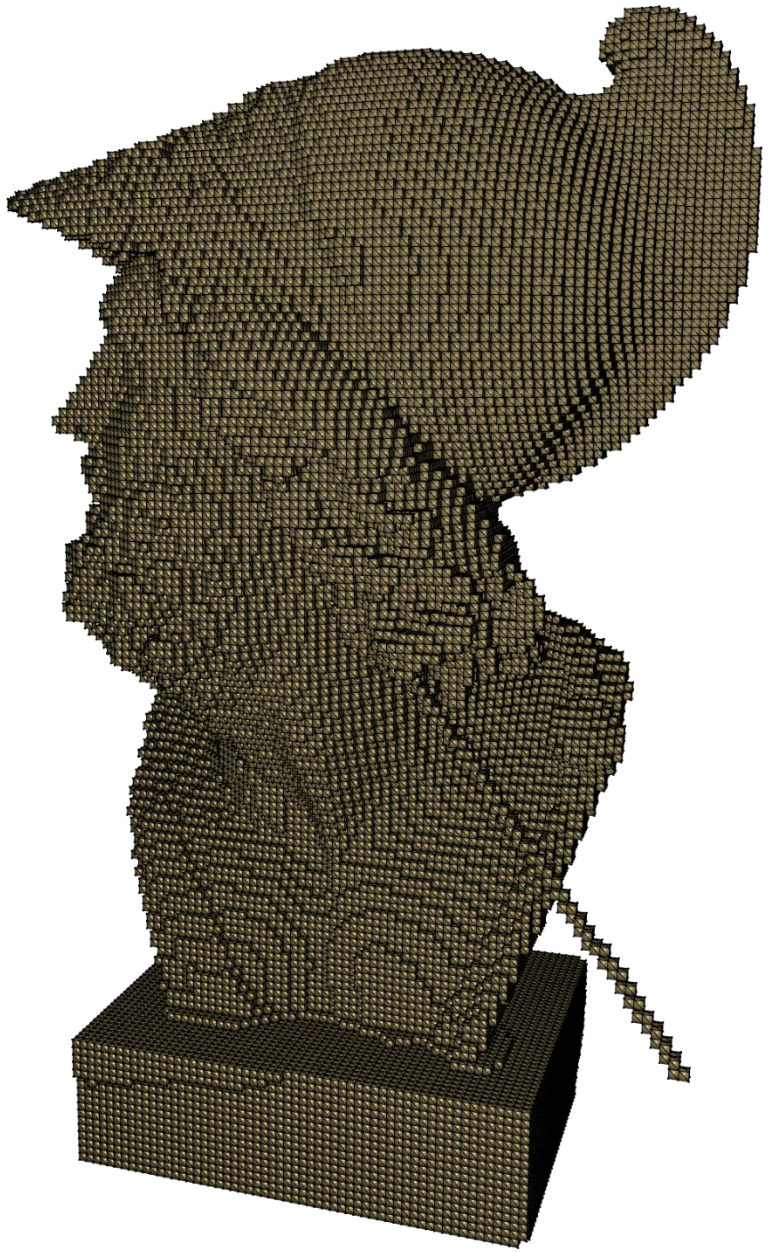


A fast voxel traversal algorithm for ray tracing

- **Algoritmo:**

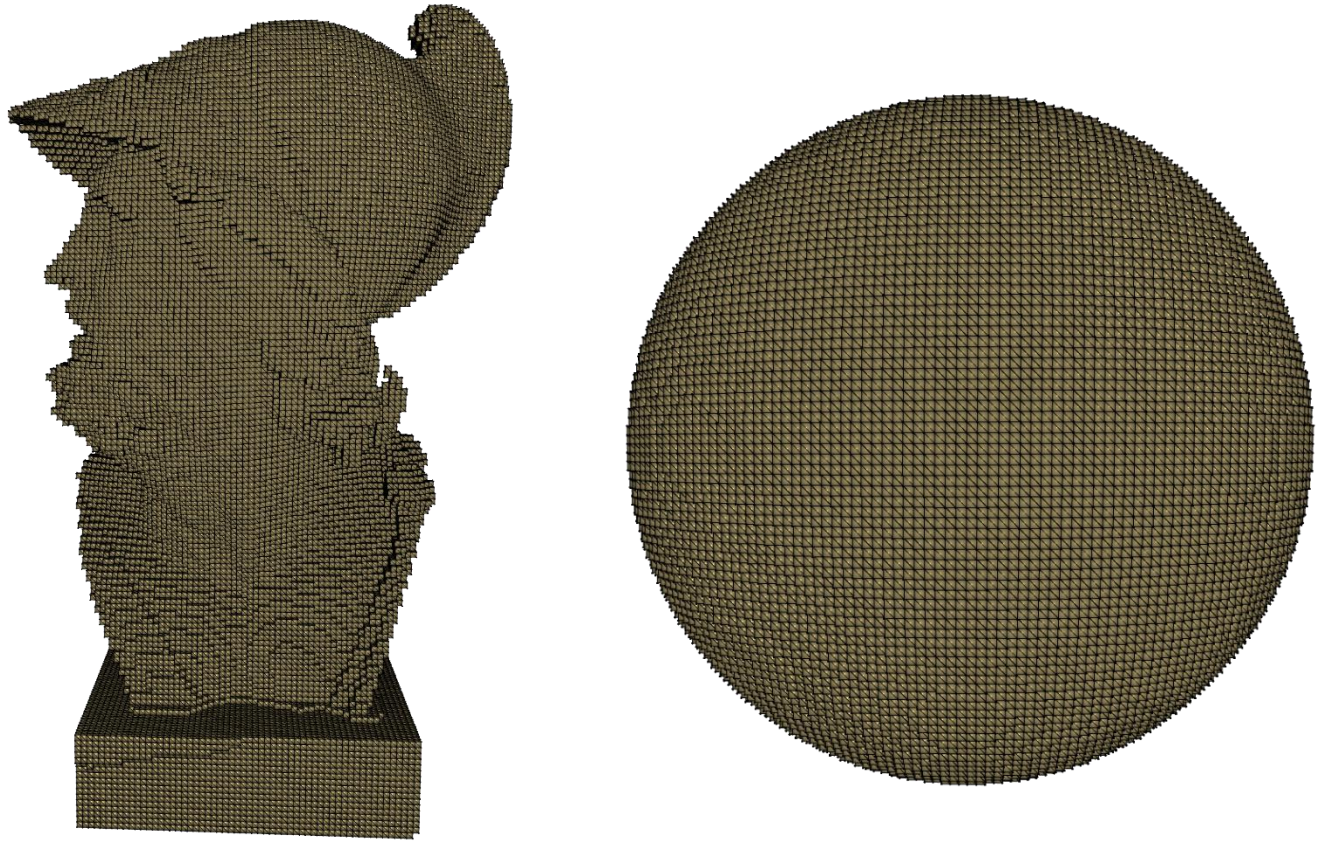
5

- Proceso iterativo.
  - Se recorre la voxelización mientras los índices sean mayores que 0 y menor que el número de subdivisiones.
  - Si `tMaxX < tMaxY y tMaxX < tMaxZ`:
    - `tDeltaX` se suma a `tMaxX`.
    - `stepX` se suma a `i`.



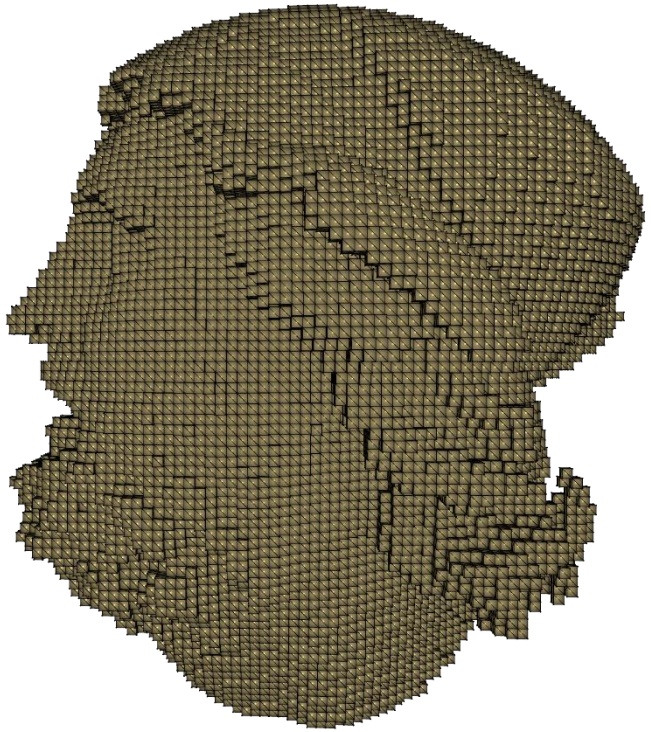
Operaciones Booleanas

- **Dada una voxelización A y una voxelización B:**
  - Una nueva voxelización que incluye la caja envolvente de A y B, con el tamaño de vóxel de A.

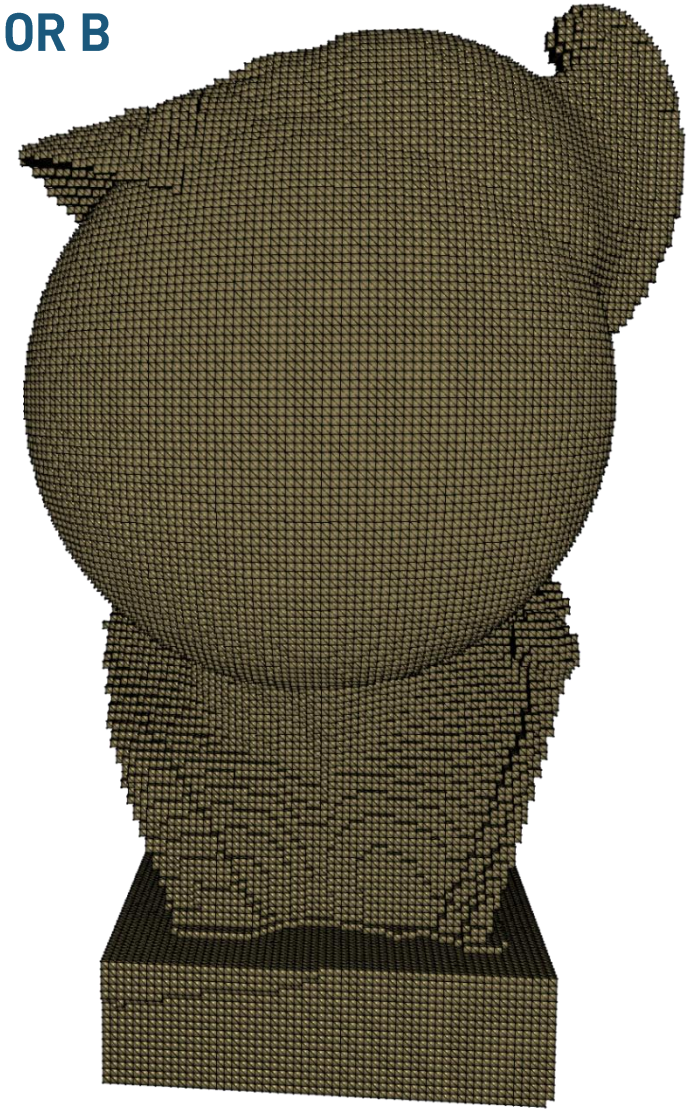


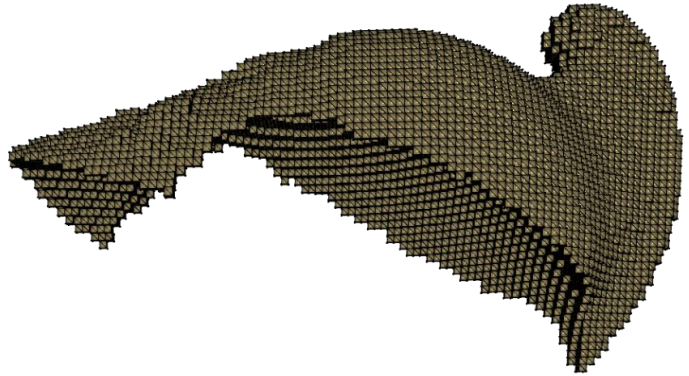
Operaciones Booleanas

A AND B

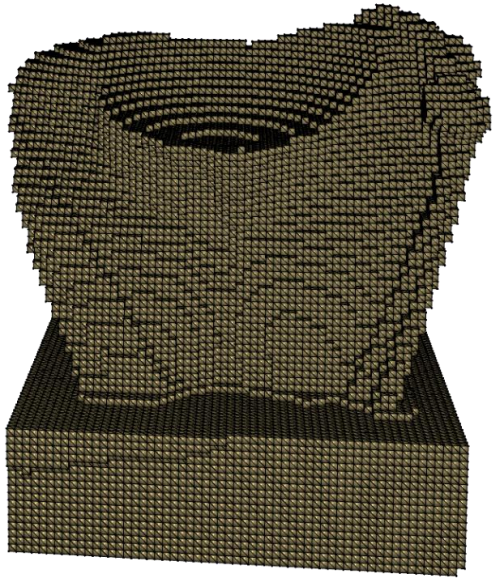


A OR B





A XOR B



Universidad de Jaén

Departamento de Informática

EFQM
Recognised for Excellence
5 Star - 2017

Práctica 5.A

Algoritmos geométricos

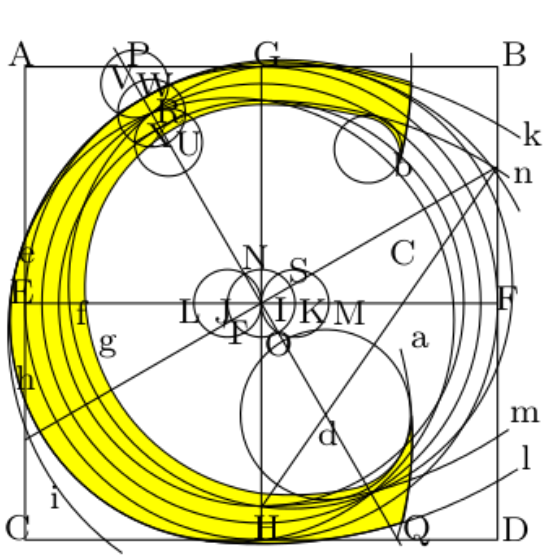
Grado en Ingeniería Informática

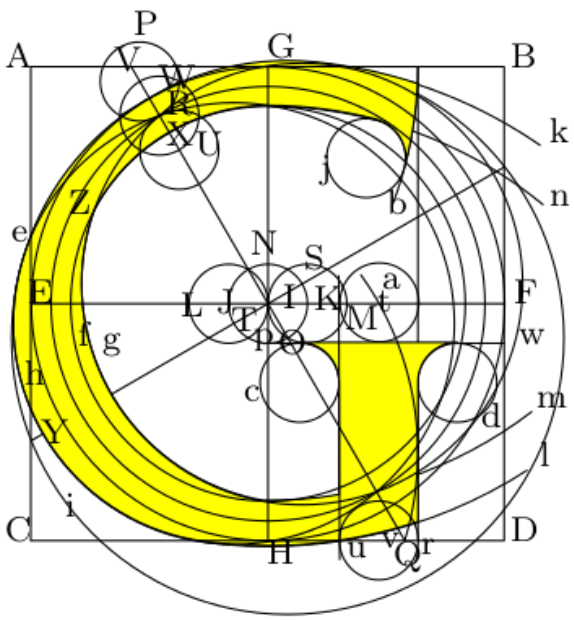
Alfonso López Ruiz

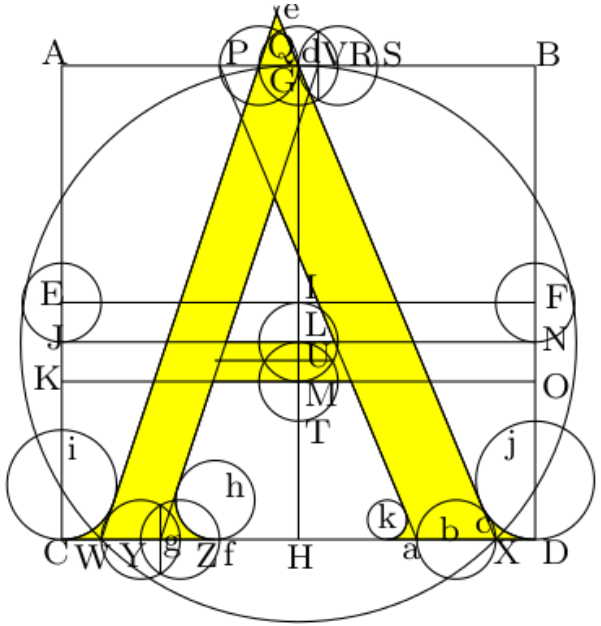
Curso 2022-2023

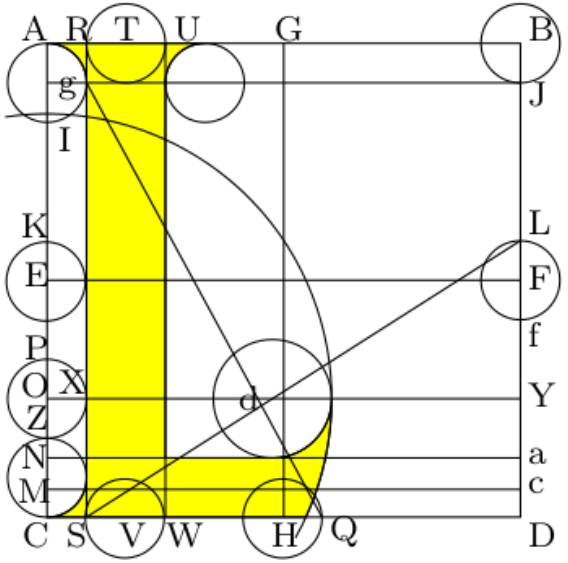
CGAL

- **The Computational Geometry Algorithms Library**
  - Procesamiento de nubes de puntos.
  - Procesamiento de mallas de triángulos.
  - Sintaxis compleja.
    - `typedef` para simplificarla



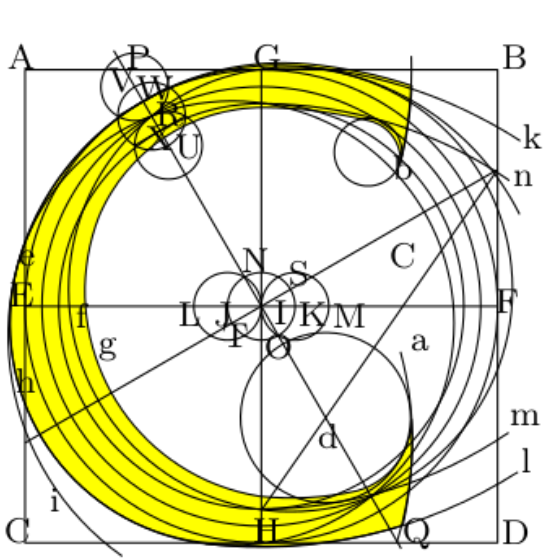


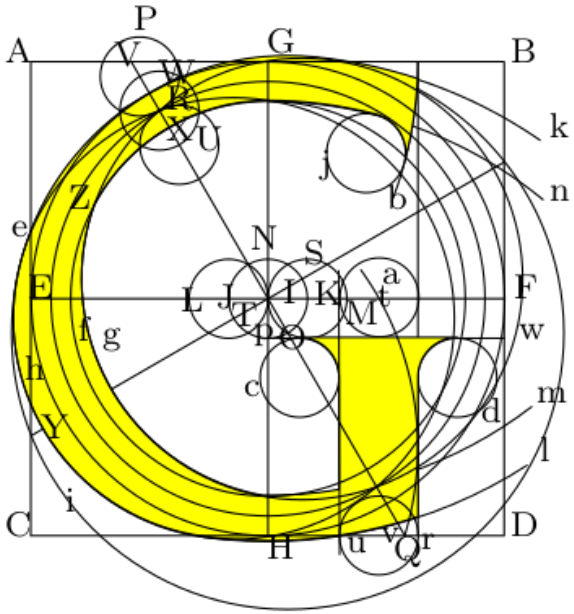


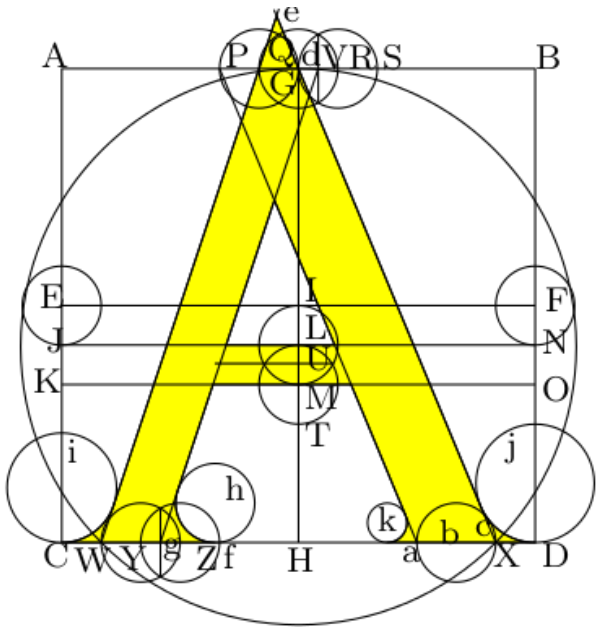


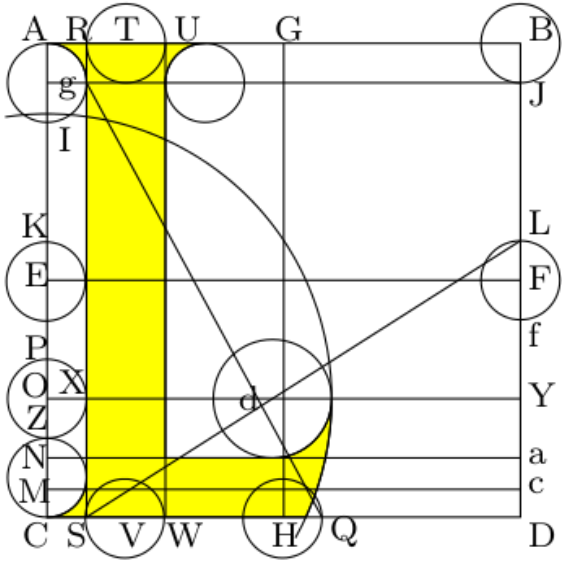
CGAL

- **The Computational Geometry Algorithms Library**
  - Sintaxis compleja.
    - `CGAL::Delaunay_triangulation_2<CGAL::Projection_traits_xy_3< CGAL::Exact_predicates_inexact_constructions_kernel>>`



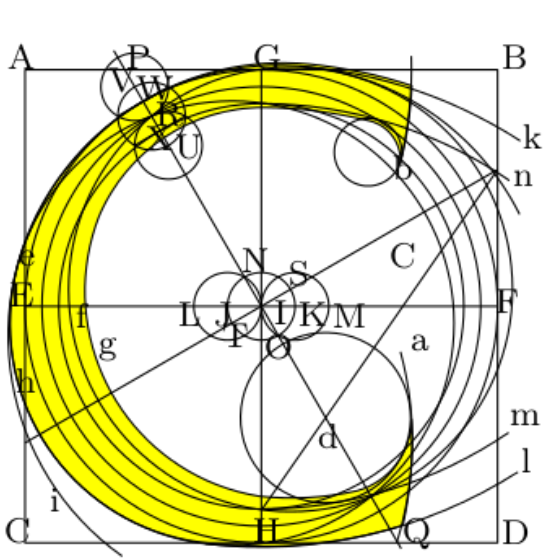


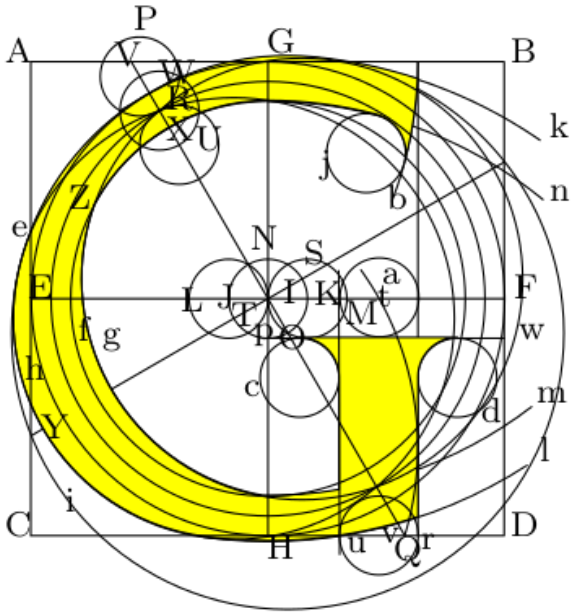


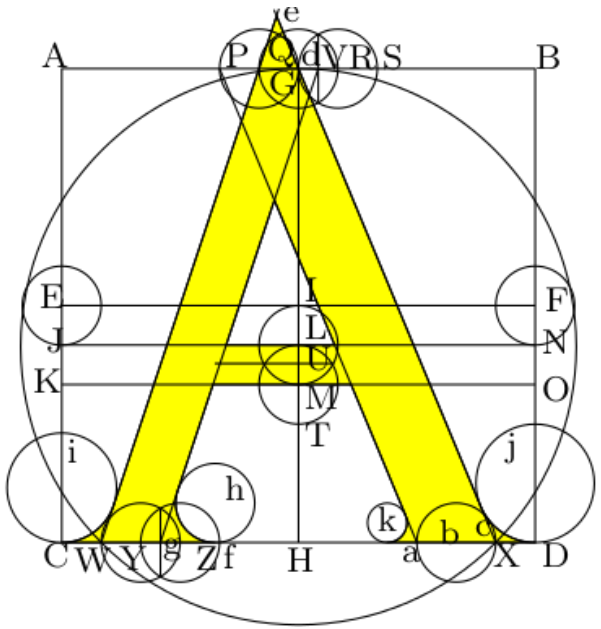


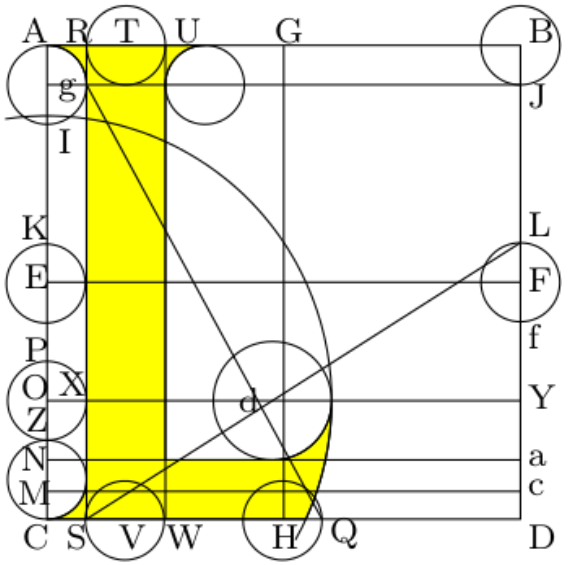
CGAL

- **The Computational Geometry Algorithms Library**
  - Sintaxis compleja.
    - `typedef CGAL::Exact_predicates_inexact_constructions_kernel K;`
    - `typedef CGAL::Projection_traits_xy_3<K> Gt;`
    - `typedef CGAL::Delaunay_triangulation_2<Gt> Delaunay;`
    - `typedef K::Point_3 Point;`



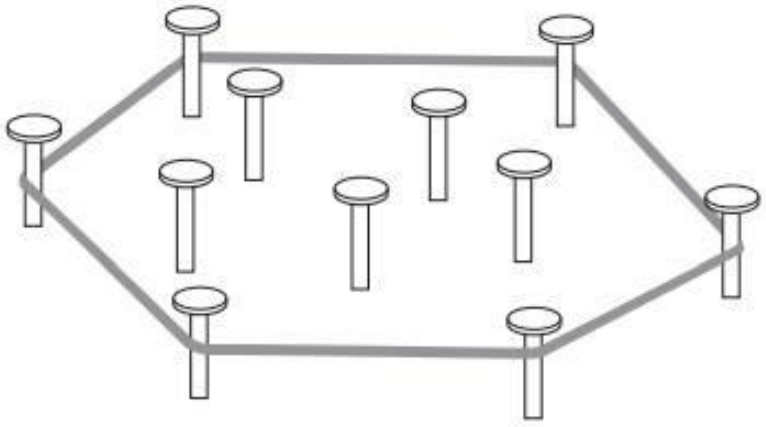


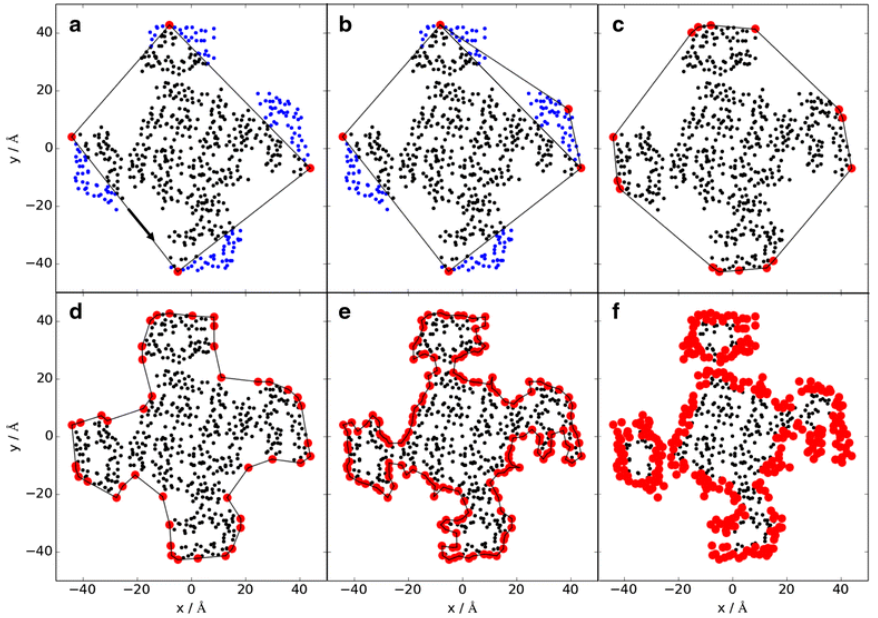




Envolvente convexa

- **La envolvente convesa encapsula un conjunto de puntos.**
  - Obtiene un envoltorio en forma de malla de triángulos (3D).
  - La envolvente convexa es mucho más estricta que la cóncava.
    - Se adapta peor a la forma de los puntos.
    - Aporta una representación mucho más fácil en la que se pueden llevar a cabo comprobaciones con menor tiempo de respuesta.





CGAL

Ejemplo de uso de CGAL

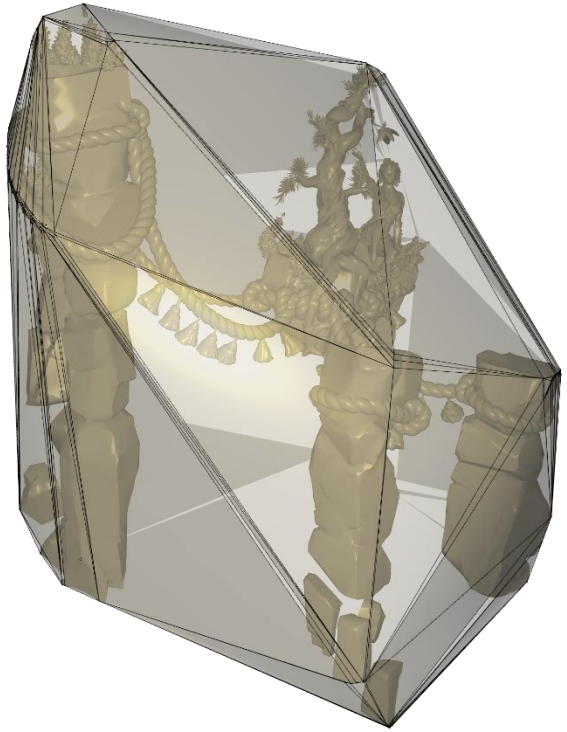
```
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Projection_traits_xy_3.h>
#include <CGAL/Delaunay_triangulation_2.h>
#include <fstream>

typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
typedef CGAL::Projection_traits_xy_3<K> Gt;
typedef CGAL::Delaunay_triangulation_2<Gt> Delaunay;
typedef K::Point_3 Point;

int main()
{
      std::ifstream in("data/terrain.cin");
      std::istream_iterator<Point> begin(in);
      std::istream_iterator<Point> end;

      Delaunay dt(begin, end);
      std::cout << dt.number_of_vertices() << std::endl;
      return 0;
}
```

github.com/CGAL/cgal/

Envolvente convexa

- **Algoritmo:**

1. Convex hull



Envolvente convexa

- **Algoritmo:**

2 Triangulación con Delaunay + Convexa