



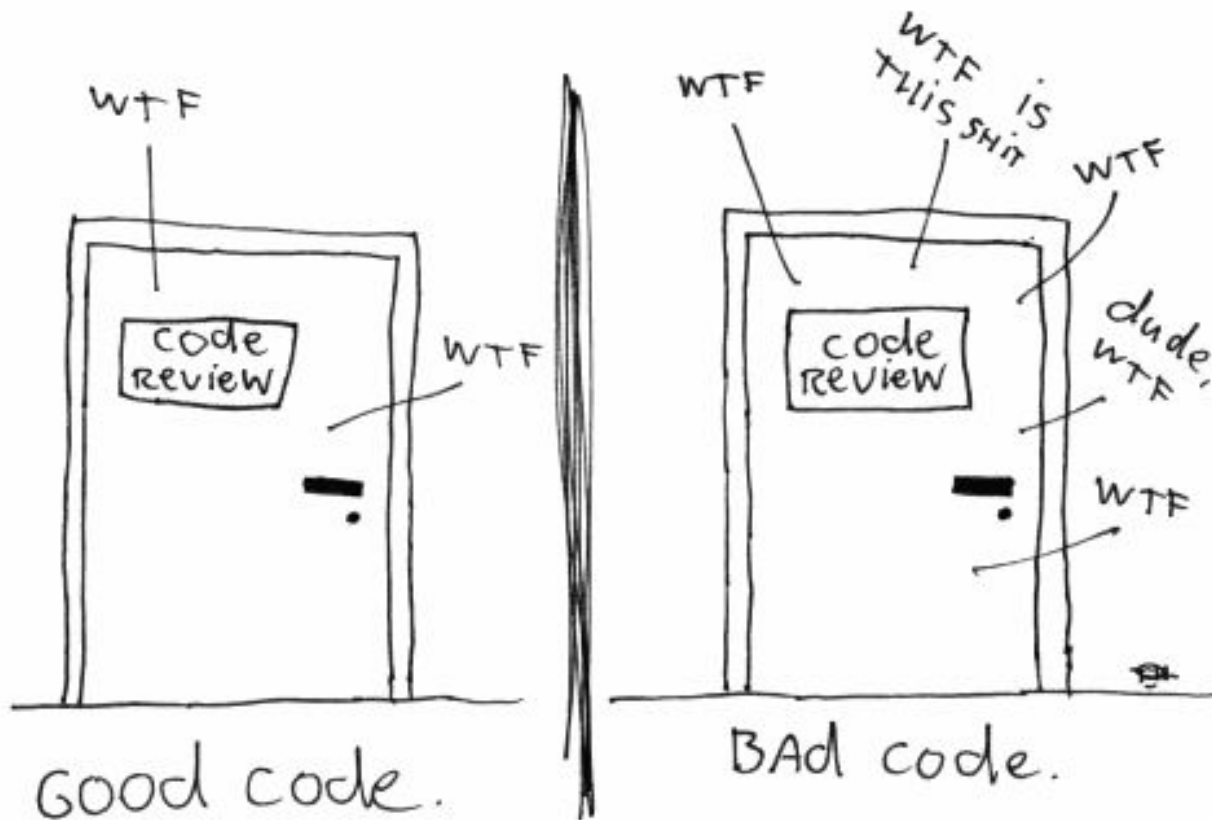
Stories of CDT

Artem Amirkhanov

Hi, I'm Artem 🖐️

- First C++ program: first year of the Uni, 2004

The ONLY VALID MEASUREMENT
OF CODE QUALITY: WTFs/MINUTE



(c) 2008 Focus Shift

Hi, I'm Artem 🖐️

- First C++ program: first year of the Uni, 2004
- First StockholmCpp meetup: 0x0F, September 2018

Hi, I'm Artem 🖐️

- First C++ program: first year of the Uni, 2004
- First StockholmCpp meetup: 0x0F, September 2018
- I write C++ libraries for living at Leica Geosystems, 2016

Hi, I'm Artem 🖐️

- First C++ program: first year of the Uni, 2004
- First StockholmCpp meetup: 0x0F, September 2018
- I write C++ libraries for living at Leica Geosystems, 2016



Hi, I'm Artem 🖐️

- First C++ program: first year of the Uni, 2004
- First StockholmCpp meetup: 0x0F, September 2018
- I write C++ libraries for living at Leica Geosystems, 2016
- I maintain an open-source C++ library for
Constrained Delaunay Triangulation (CDT)
github.com/artem-ogre/CDT, since 2019

Expectation management

I am expected:

- Introduction to the Delaunay triangulations
- Introduction to the CDT library
- A collection of stories about CDT library
 - Technical and non-technical topics

You are expected:

- Get an idea about triangulation and CDT library features
- Reflect, learn, suggest, discuss
- Nod along or shake the head disapprovingly
- Enjoy the presentation or tell me it was boring in the survey 😊

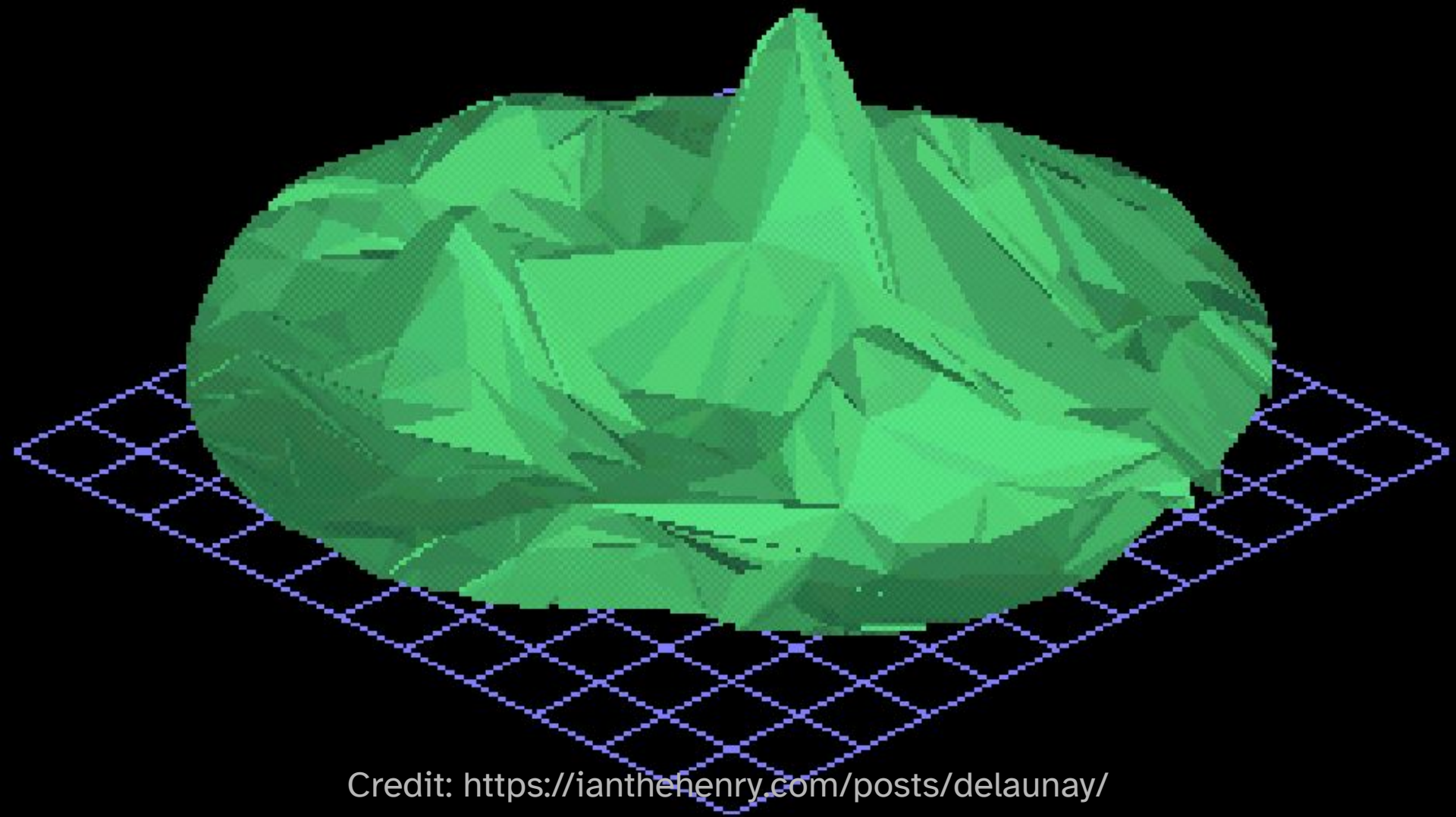
**Have you heard of CDT library
before?**



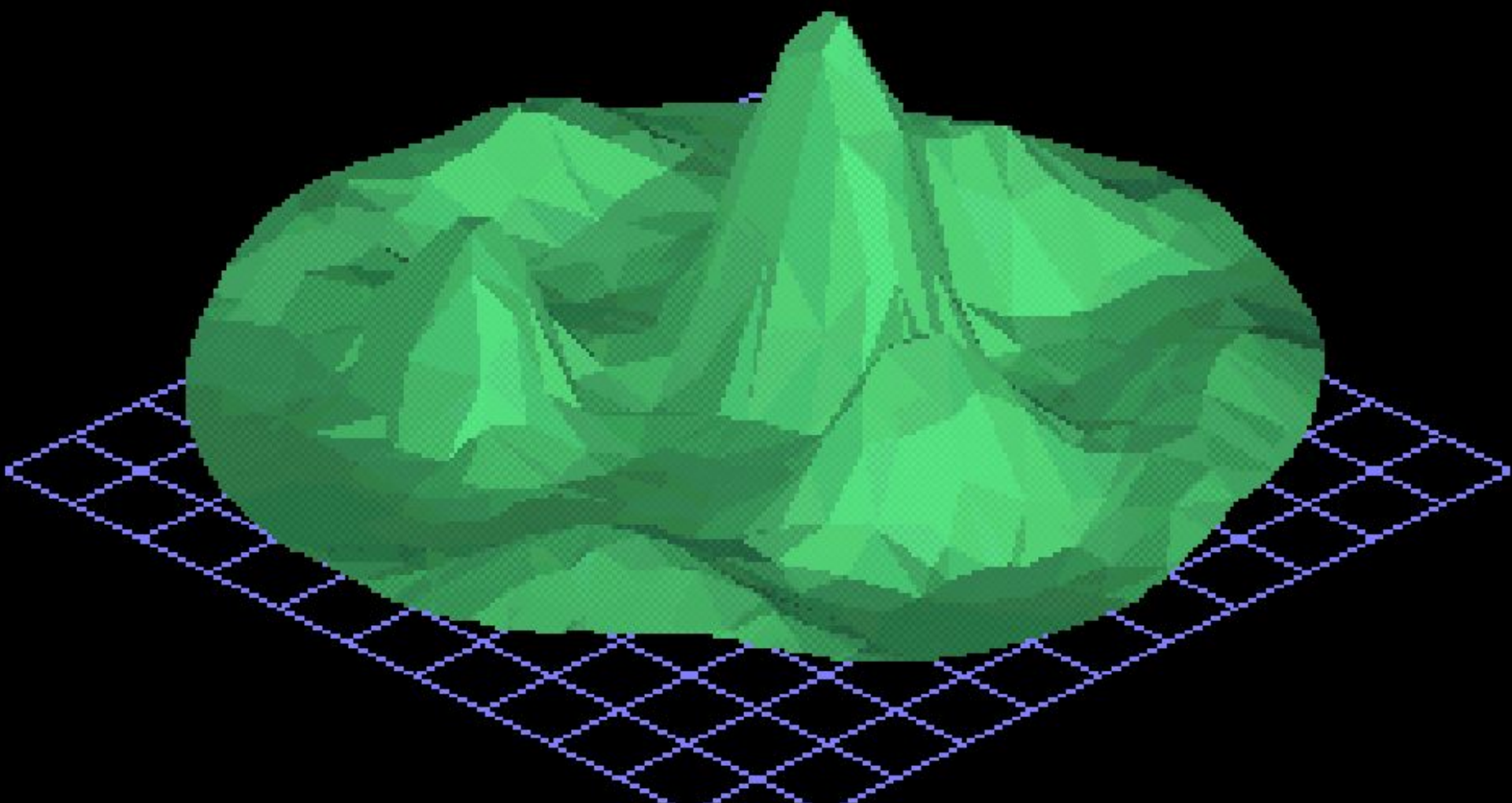
**Have you heard of Delaunay
triangulation?**







Credit: <https://ianthehenry.com/posts/delaunay/>



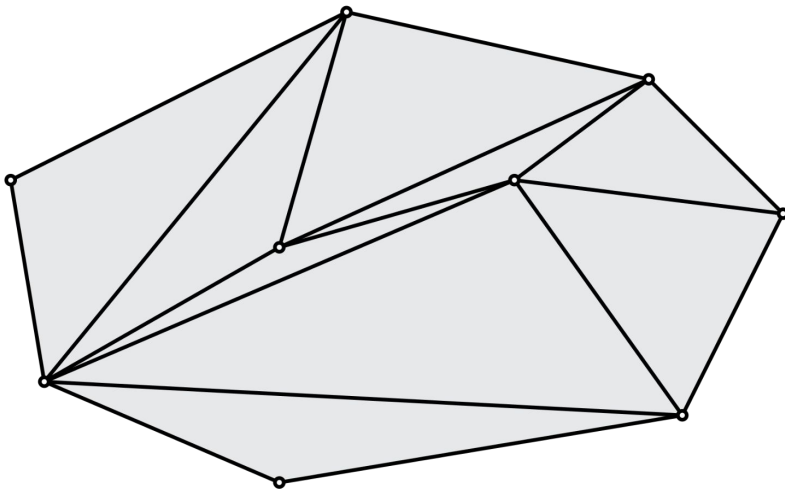
Credit: <https://ianthehenry.com/posts/delaunay/>

CDT library in a nutshell

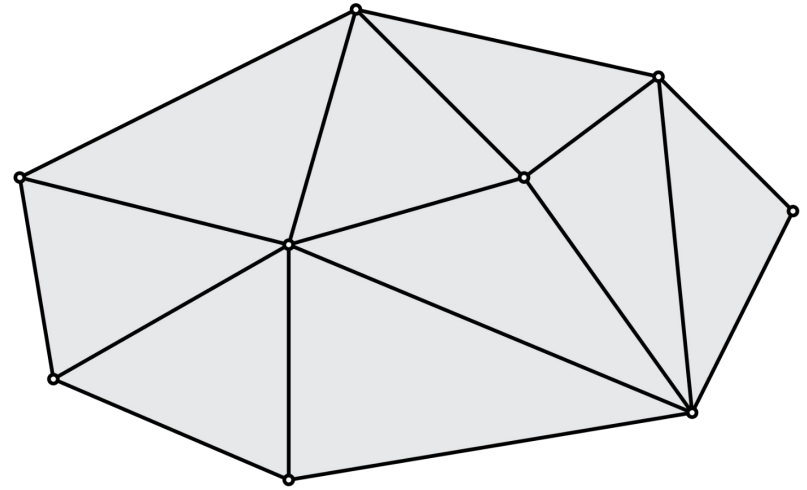
- Generates Constrained Delaunay Triangulations (CDTs)
- **Open-source:** permissively-licensed under MPL-2.0
- **Cross-platform:** tested on Windows, Linux, and macOS
- **Portable:** backwards-compatible with C++98
- **Bloat-free:** no external dependencies by default
- **Performant:** continuously profiled, measured, and optimized
- **Numerically robust:** relies on robust geometric predicates
- **Flexible:** can be a header-only or compiled

Delaunay triangulation

- **Points triangulation** is connecting 2D points into triangles
- **Delaunay triangulation** is a 'good' point triangulation
 - Maximizes the minimum angle of the triangles
 - More regular and well-shaped triangles

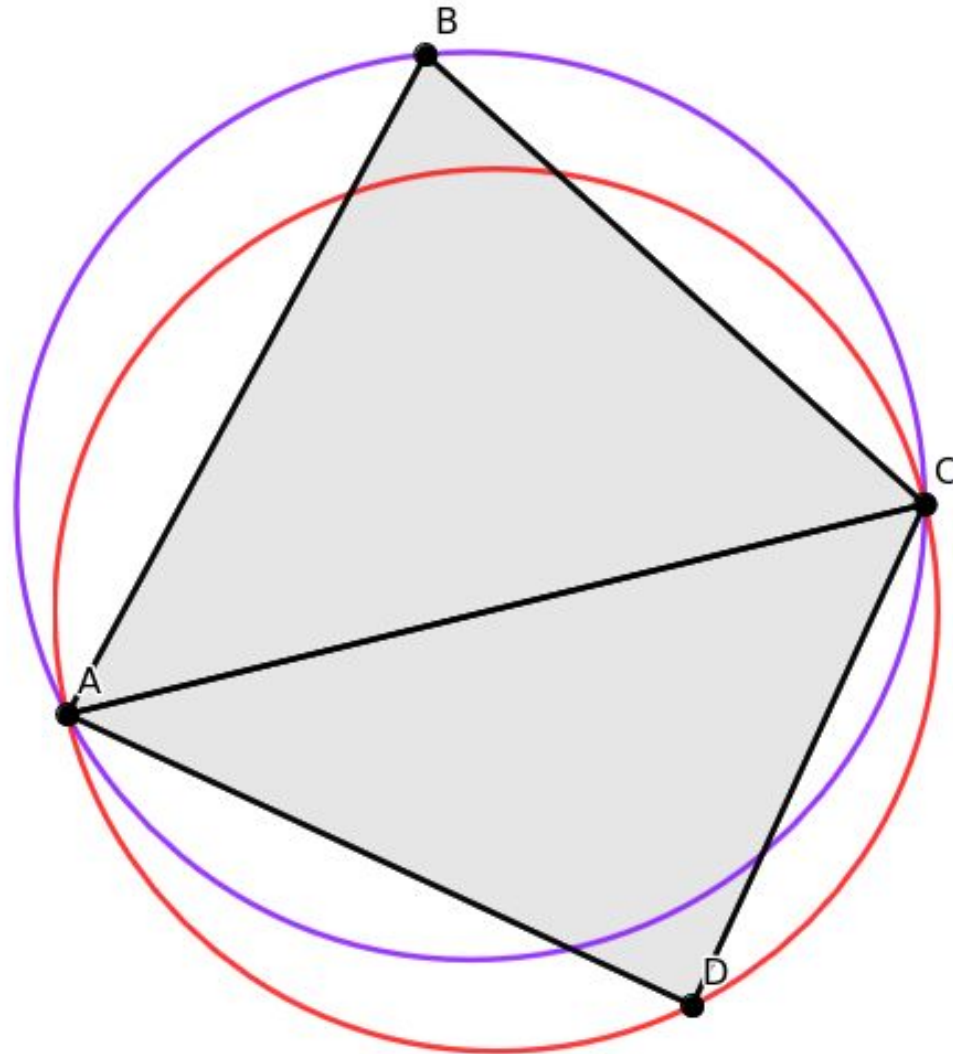


a triangulation

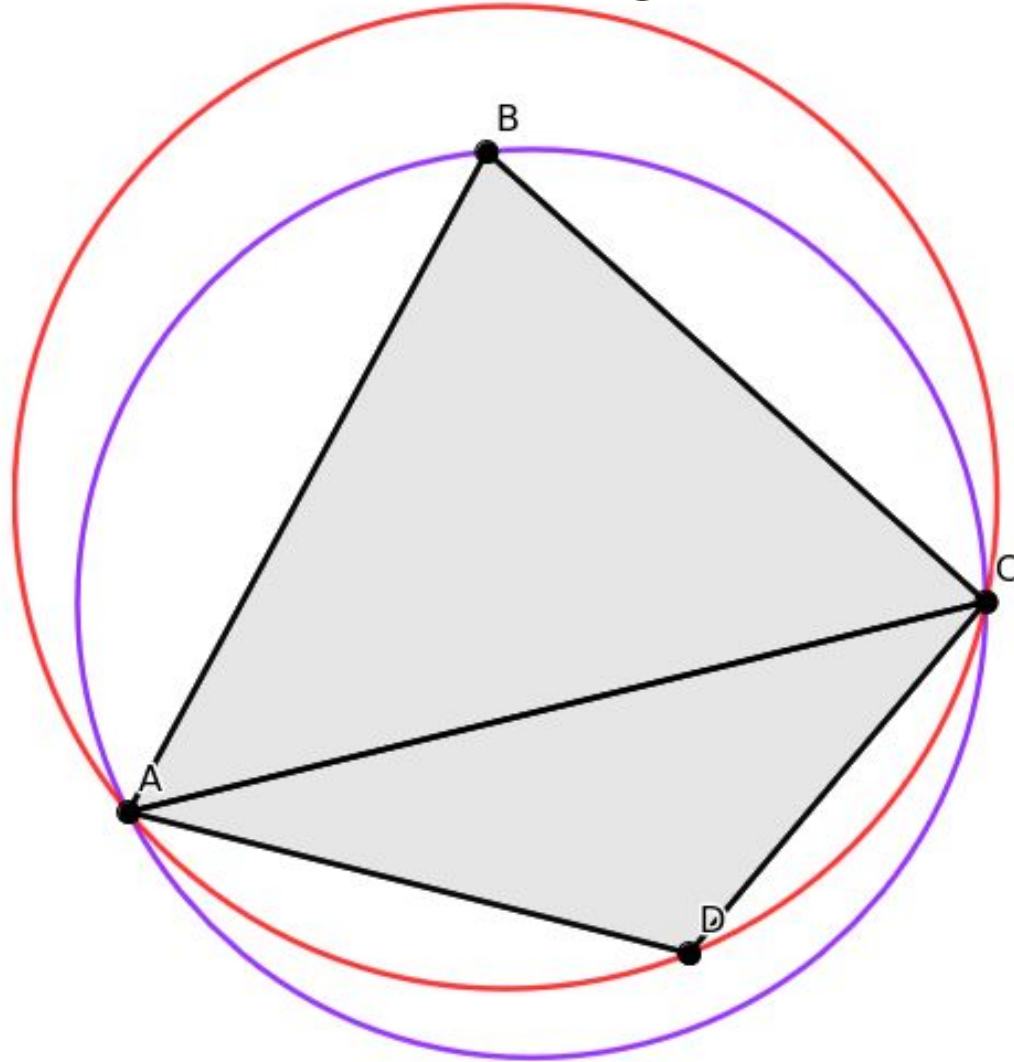


Delaunay triangulation

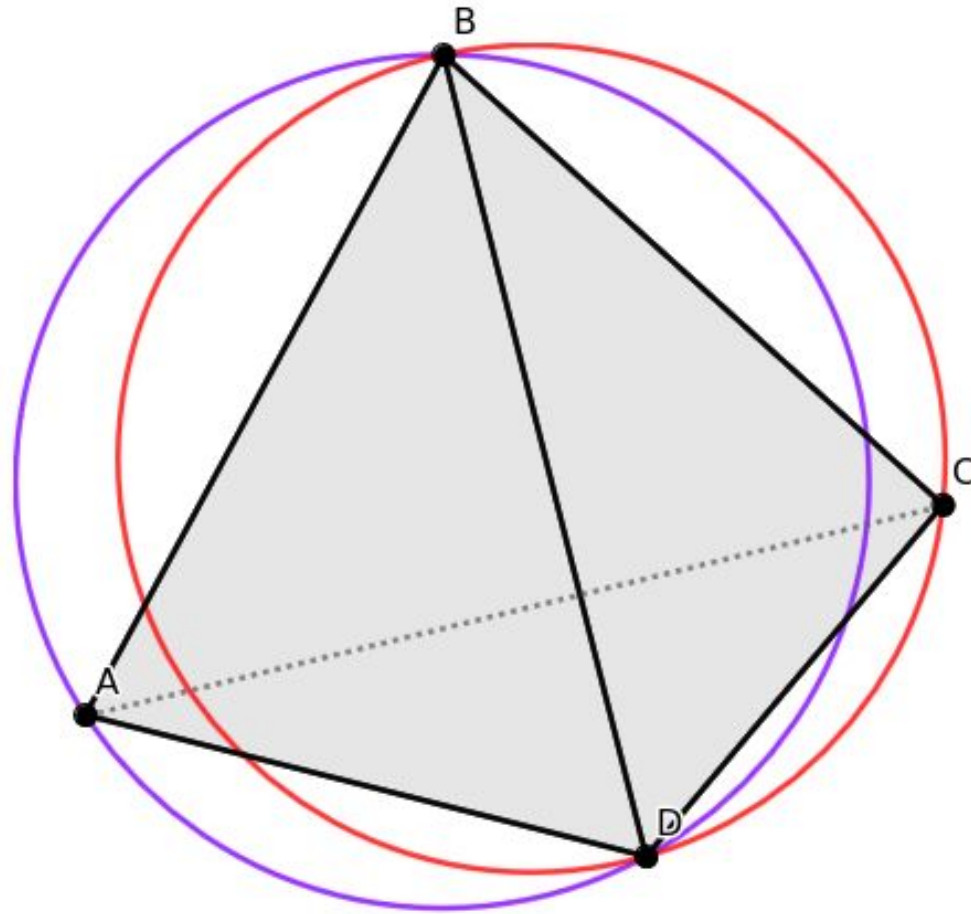
**Delaunay criterion:
circumcircle of each triangle must be empty**



**Delaunay criterion:
circumcircle of each triangle must be empty**

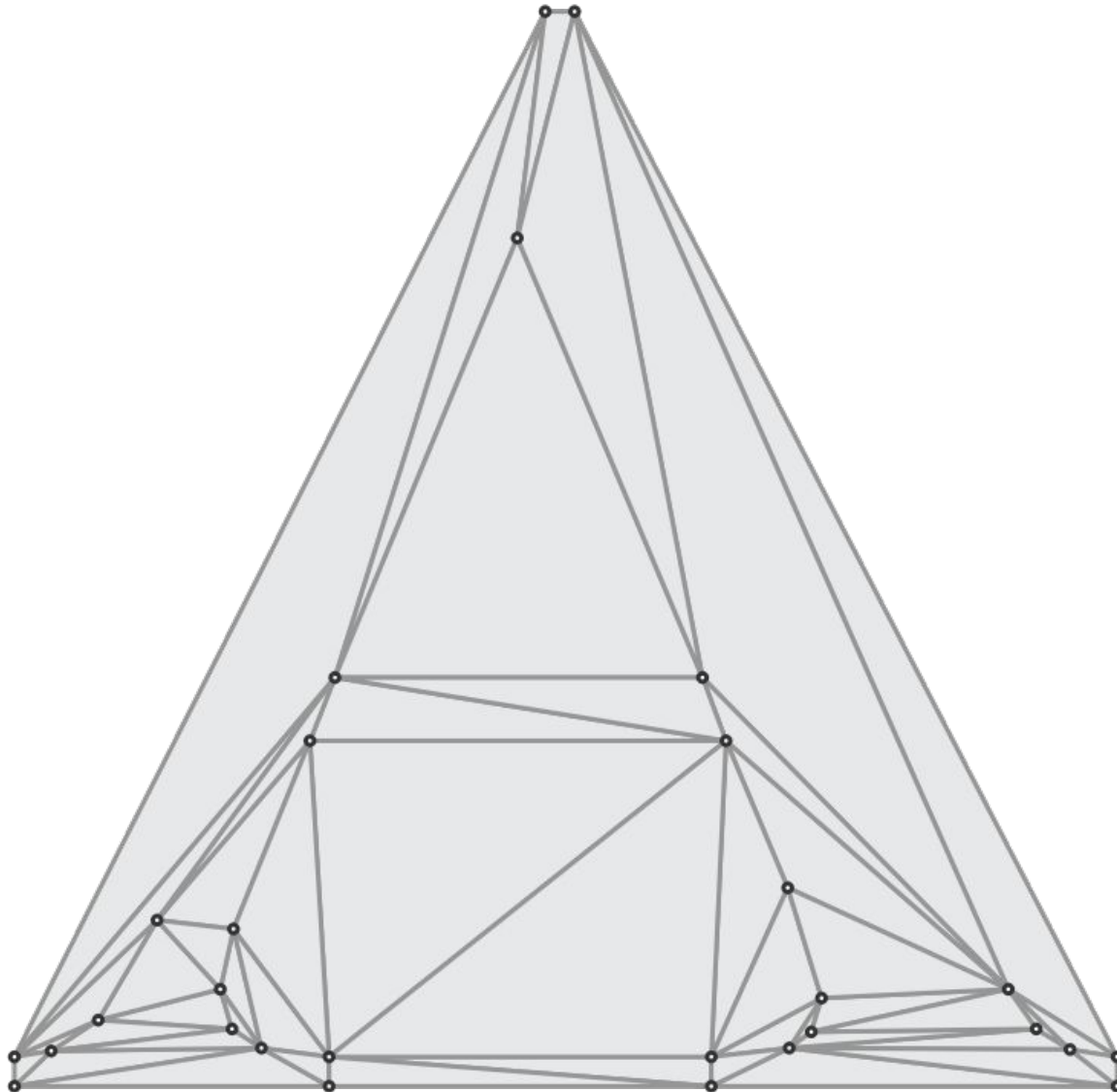


**Delaunay criterion:
circumcircle of each triangle must be empty**

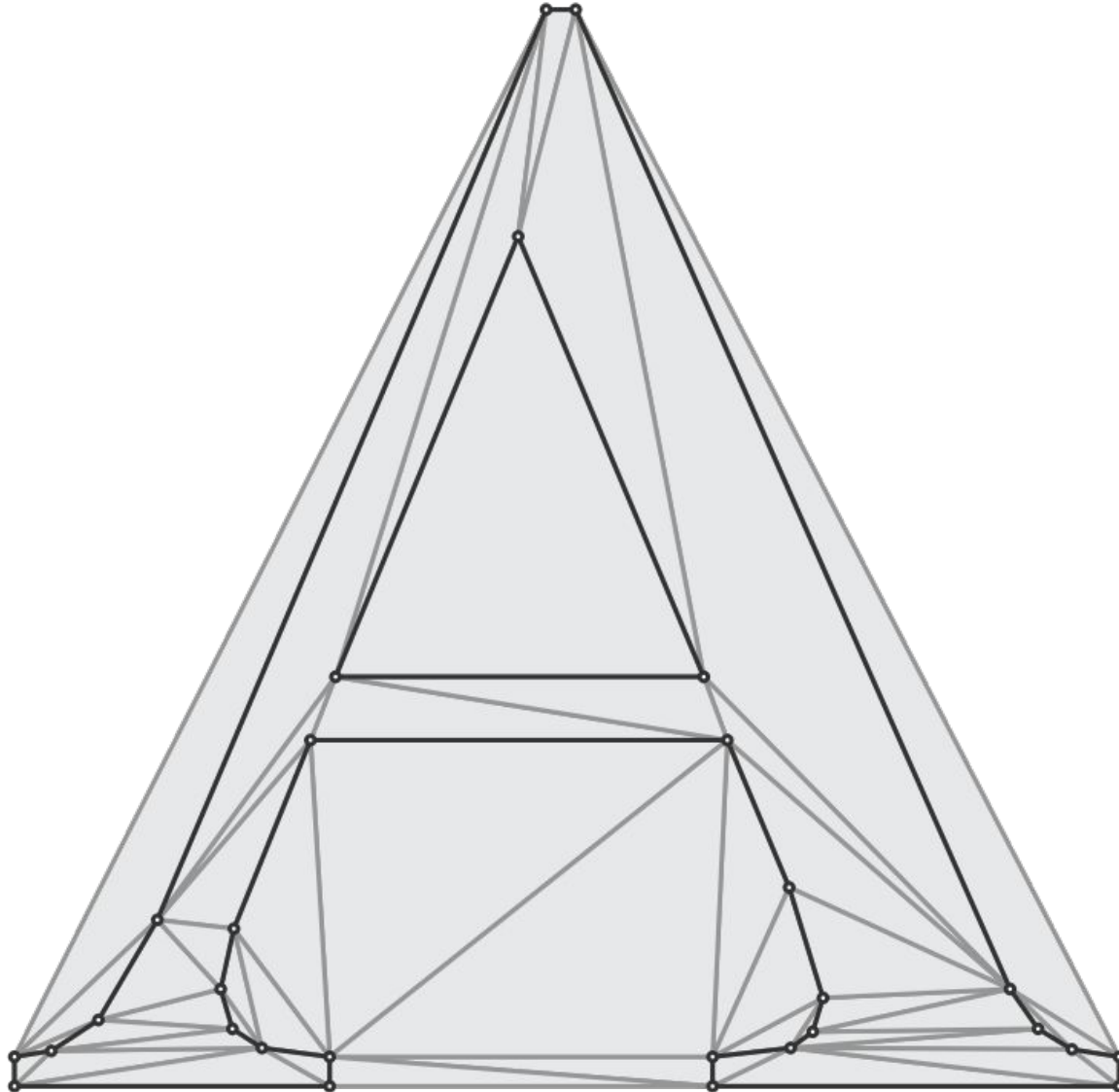


What can CDT library do

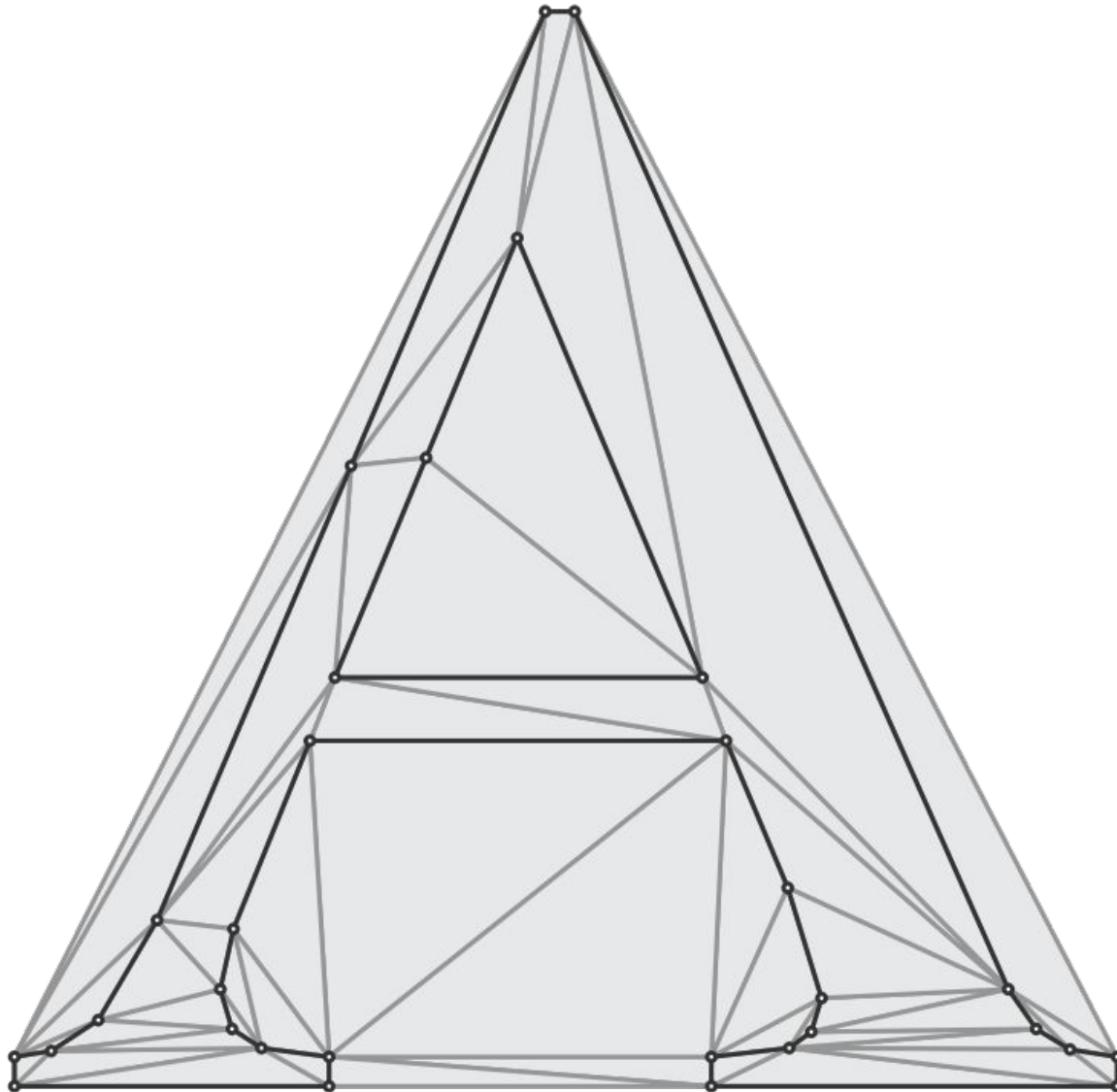
Delaunay triangulation



Constrained Delaunay triangulation



Conforming Delaunay triangulation

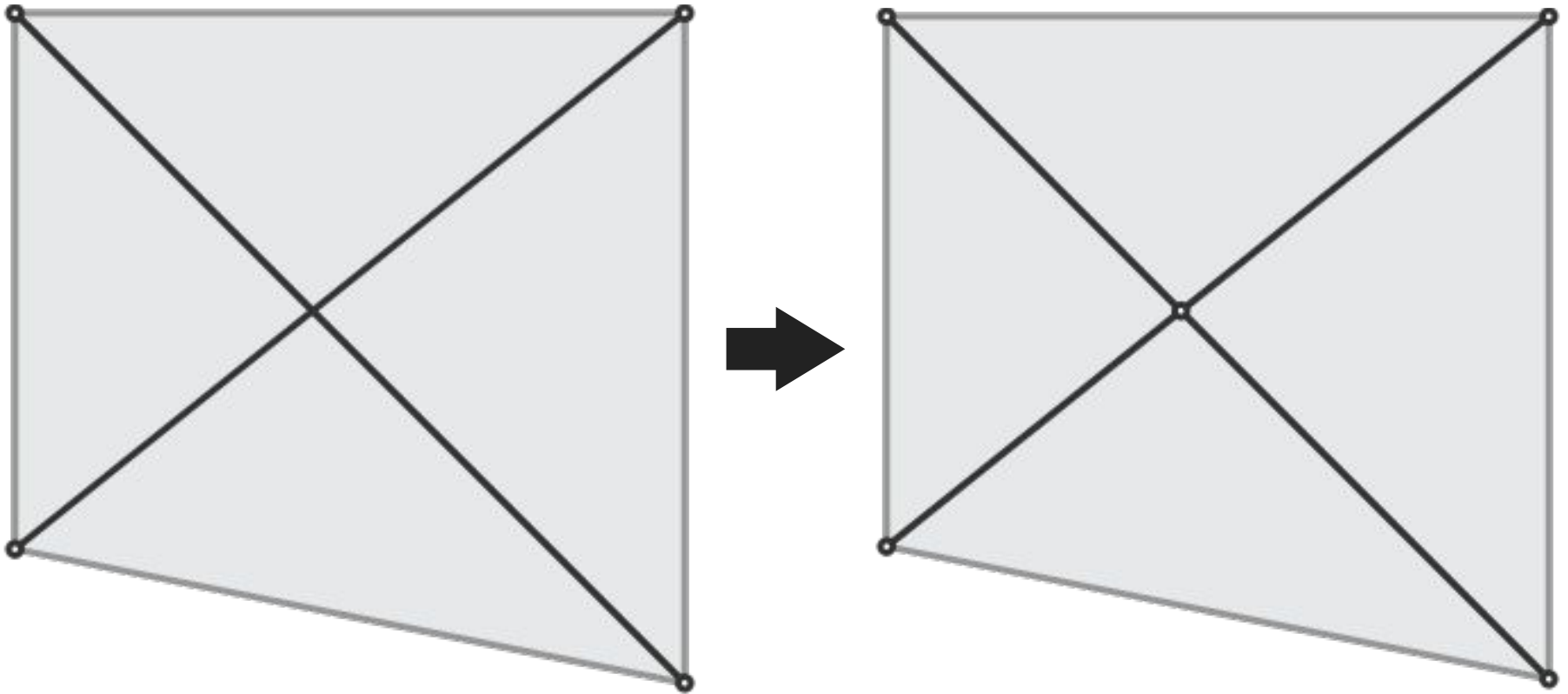


What to do if edges intersect?

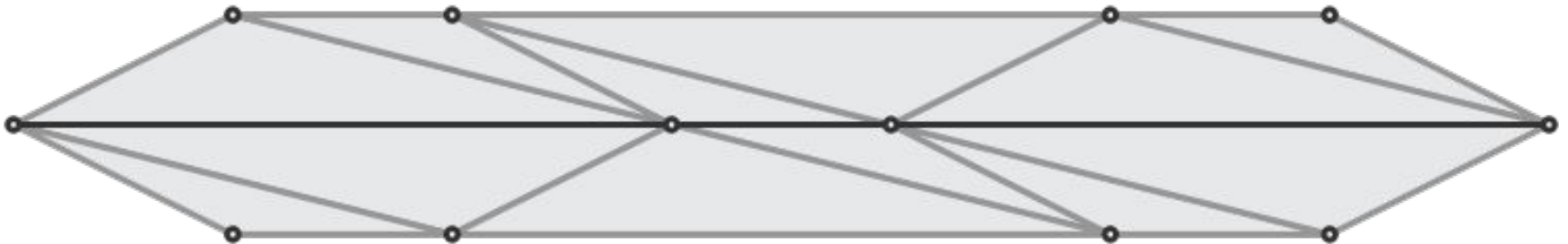
CDT lib implements three ways of dealing with edge intersections:

1. **Not allowed (default):** throw an exception with information about intersecting edges
2. **Try to resolve:** add new vertex at the edges' intersection and split the edges
3. **Don't check:** skip the checks at the user's risk, produce invalid triangulation when edges intersect

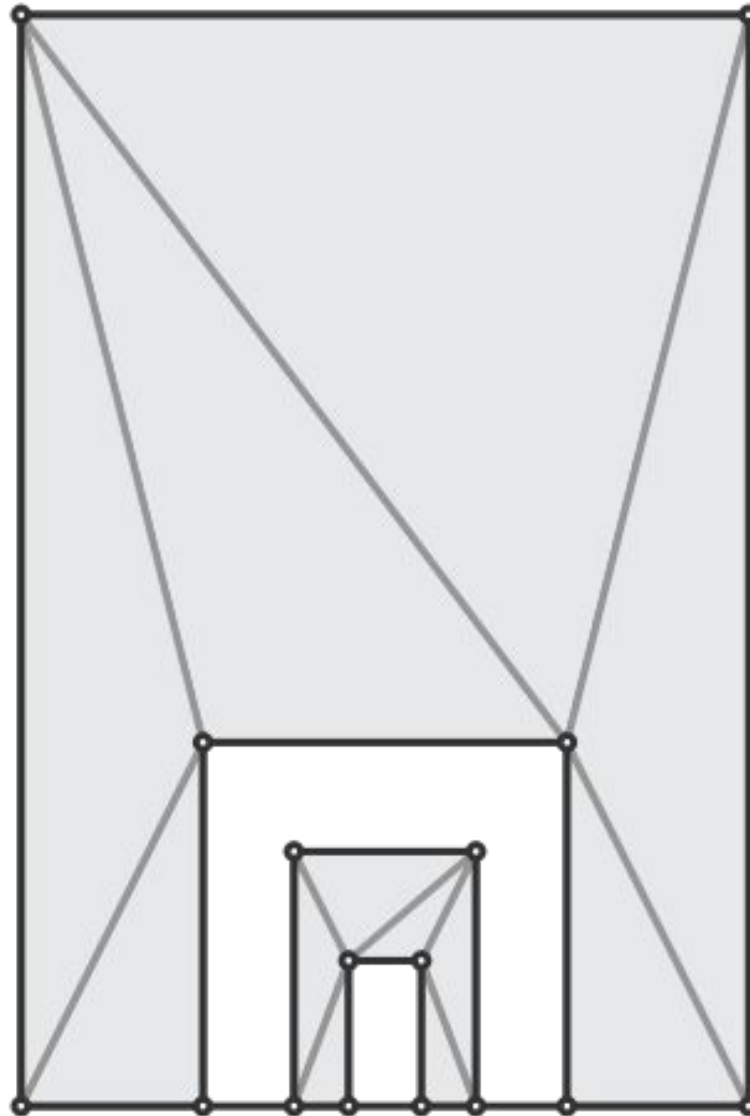
Resolve intersecting edges



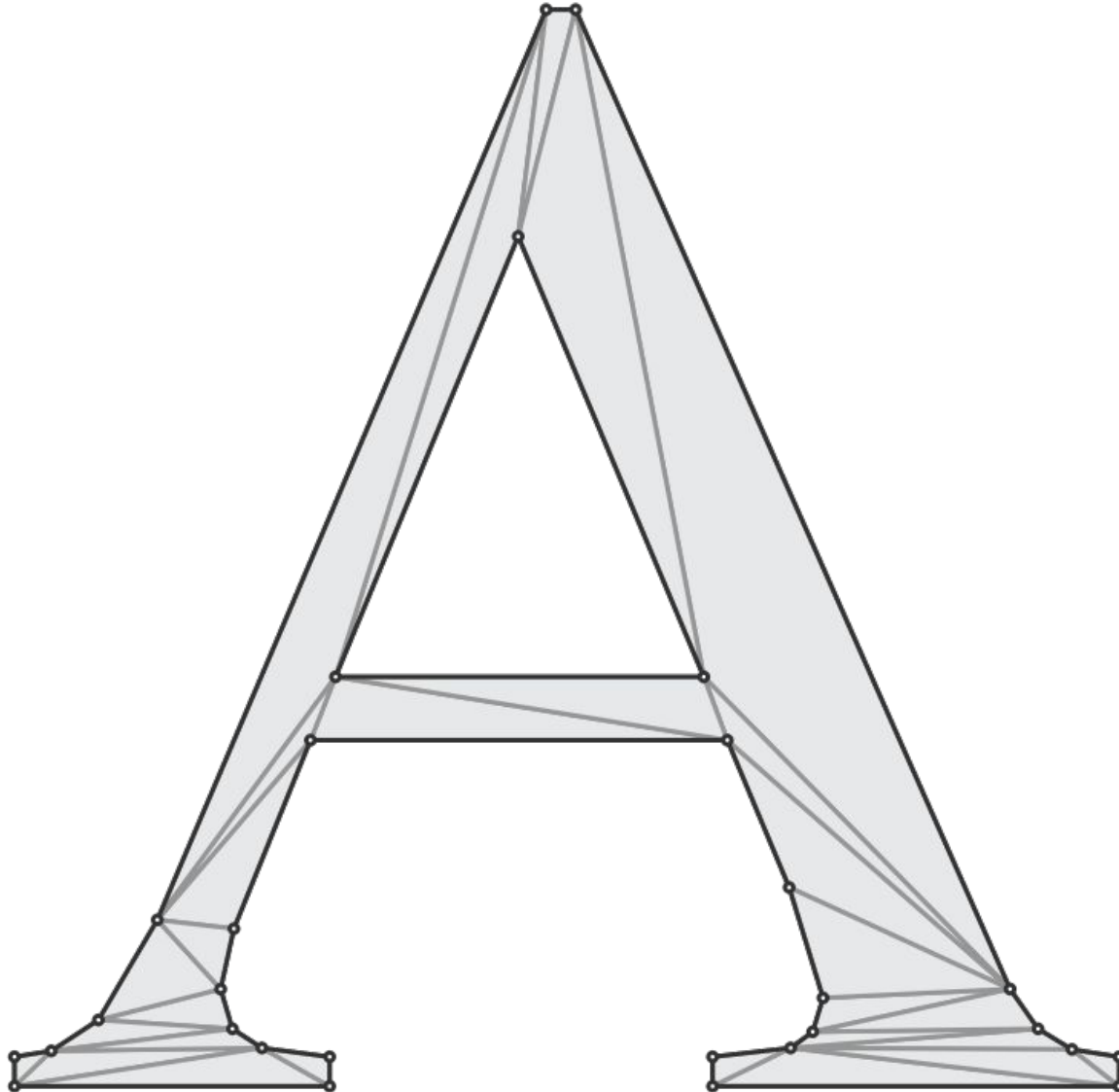
Handle points exactly on edge



Handle overlapping edges




Auto-detecting boundaries and holes



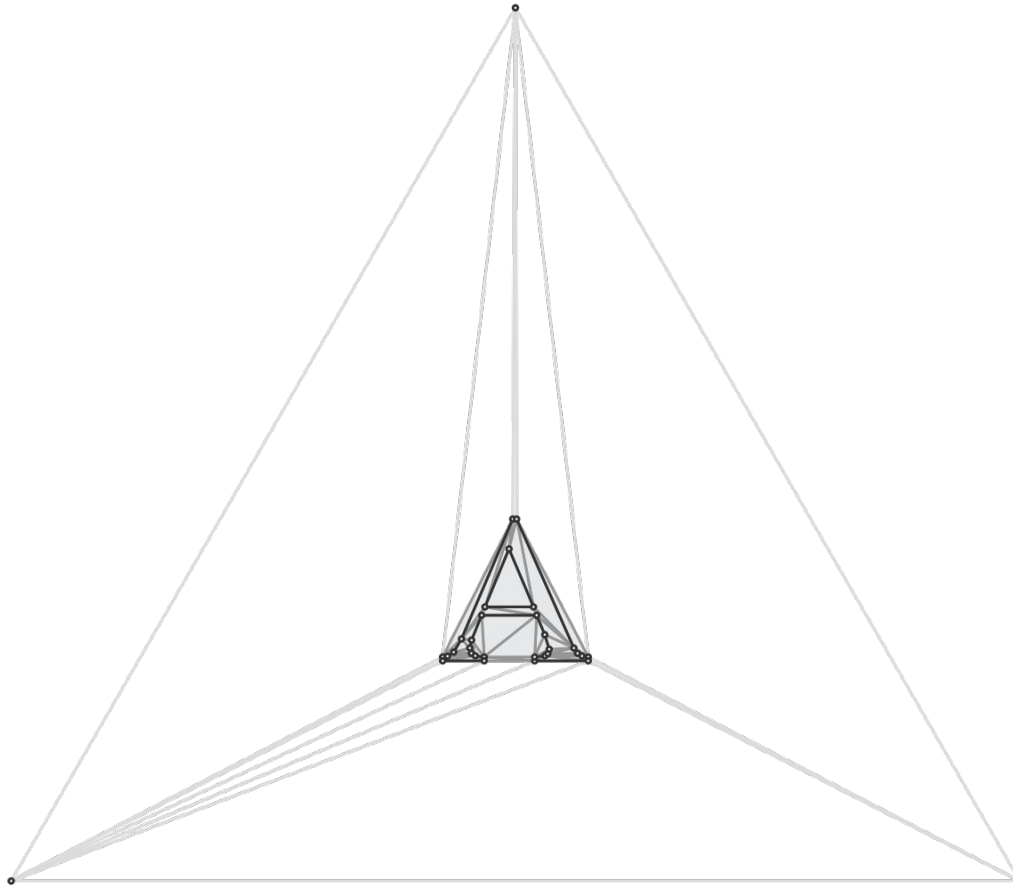
Triangulation algorithm walkthrough

Creating triangulation is an incremental algorithm

1. Start with the super-triangle 
2. Add vertices to the triangulation
3. Force edges into the triangulation
4. Finalize the triangulation: remove unnecessary triangles

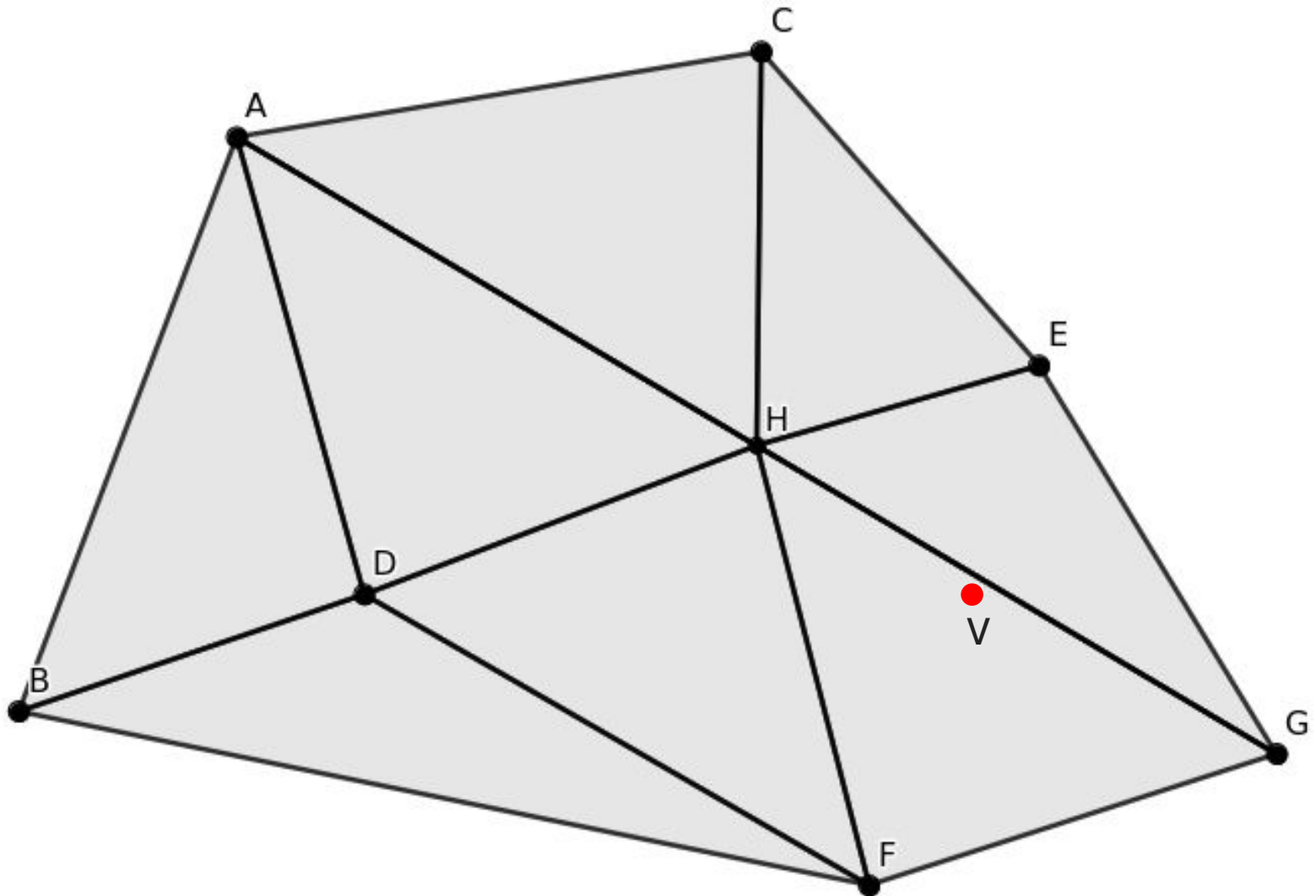
Start with a (super triangle)

- Triangle large enough to contain all the points
- Initialize triangulation with a single super triangle

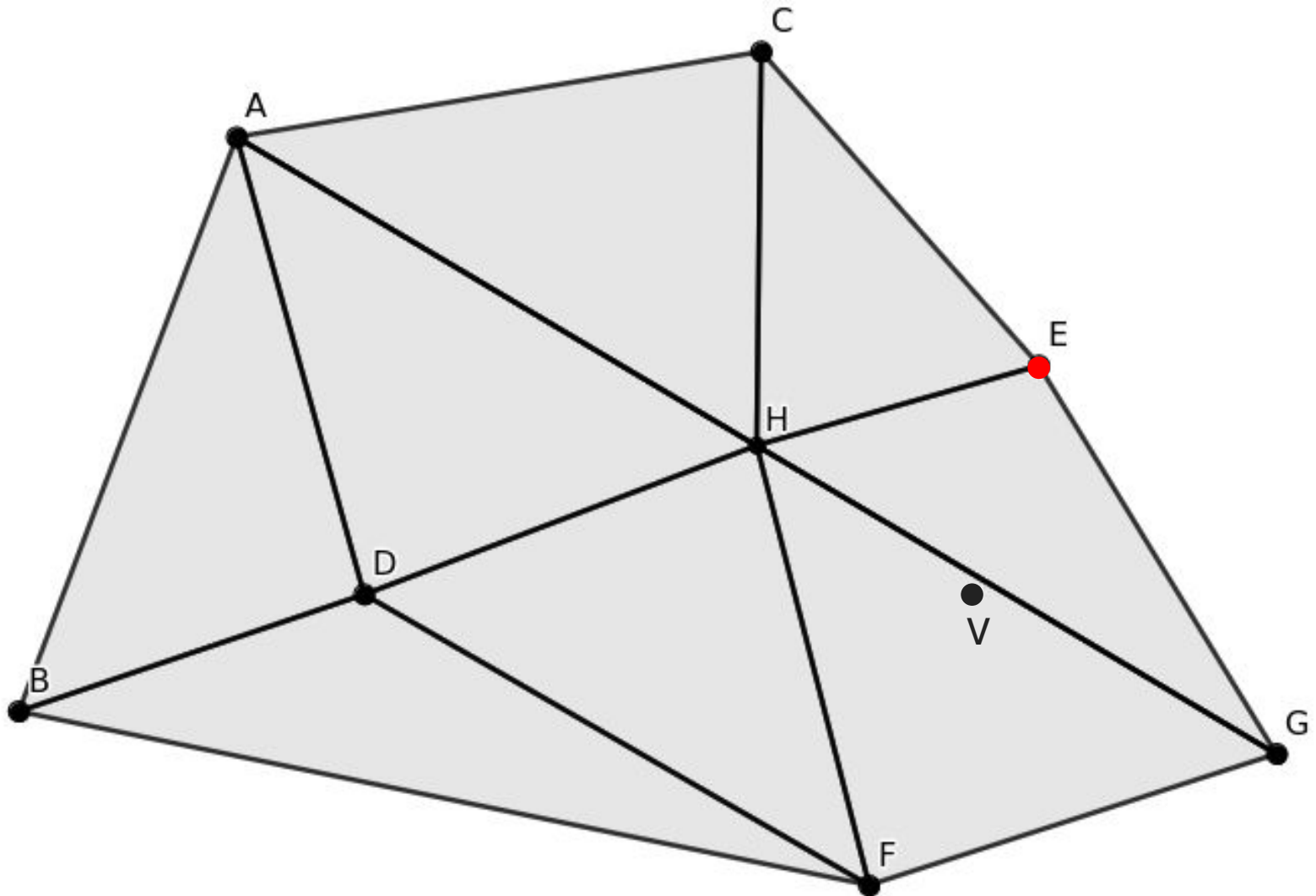


How is a vertex inserted into the triangulation?

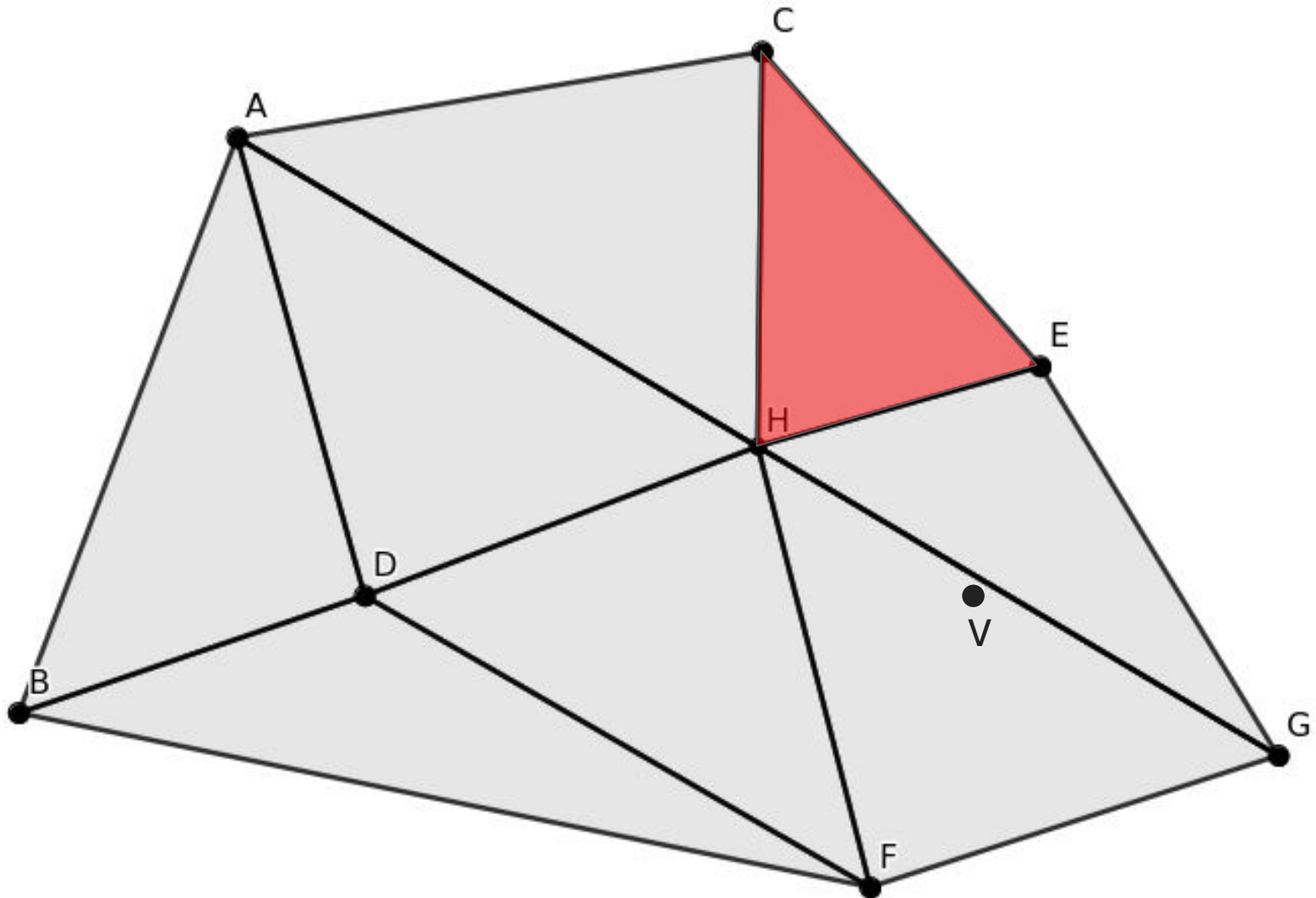
Locating new vertex



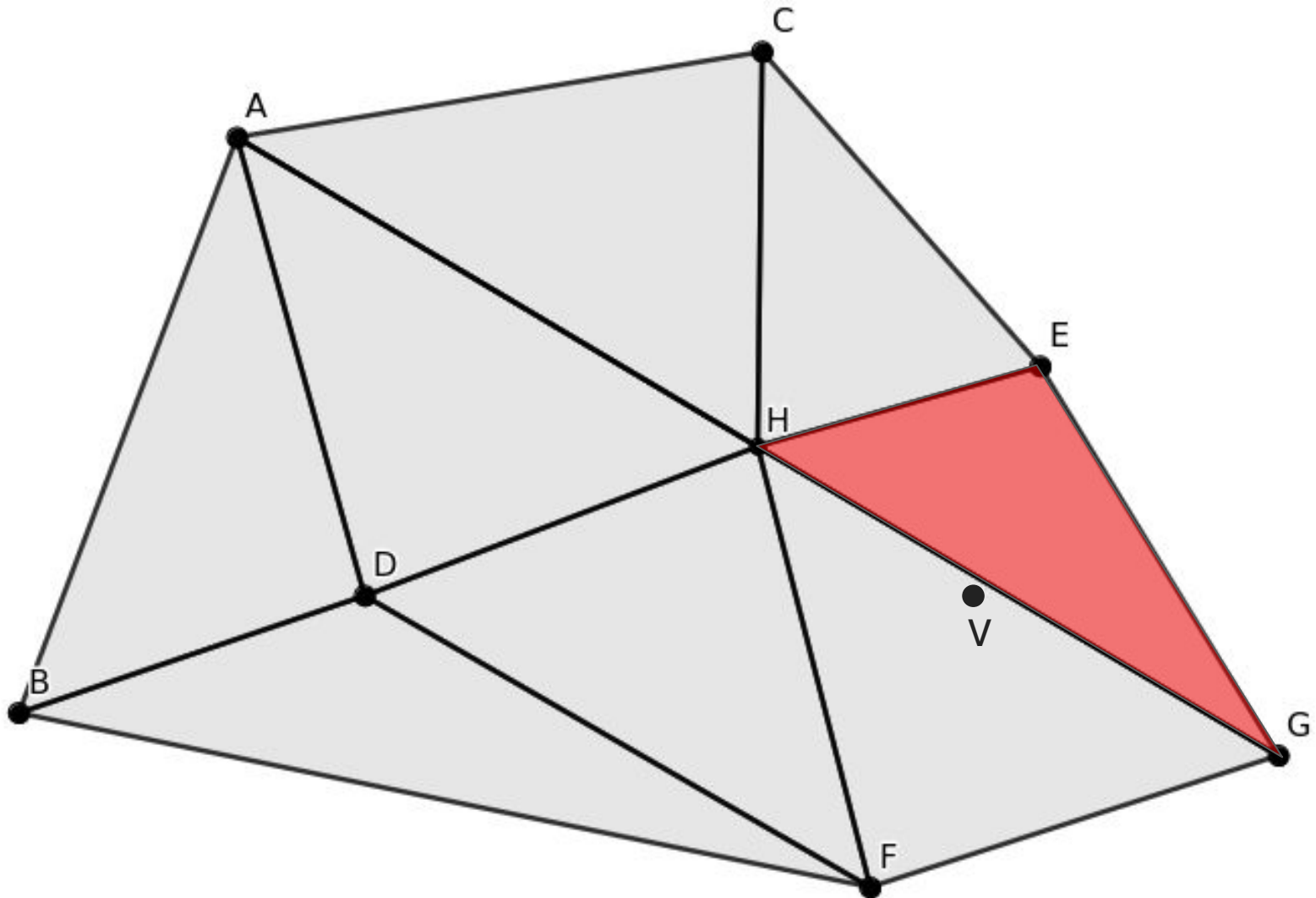
Find near(est) neighbor



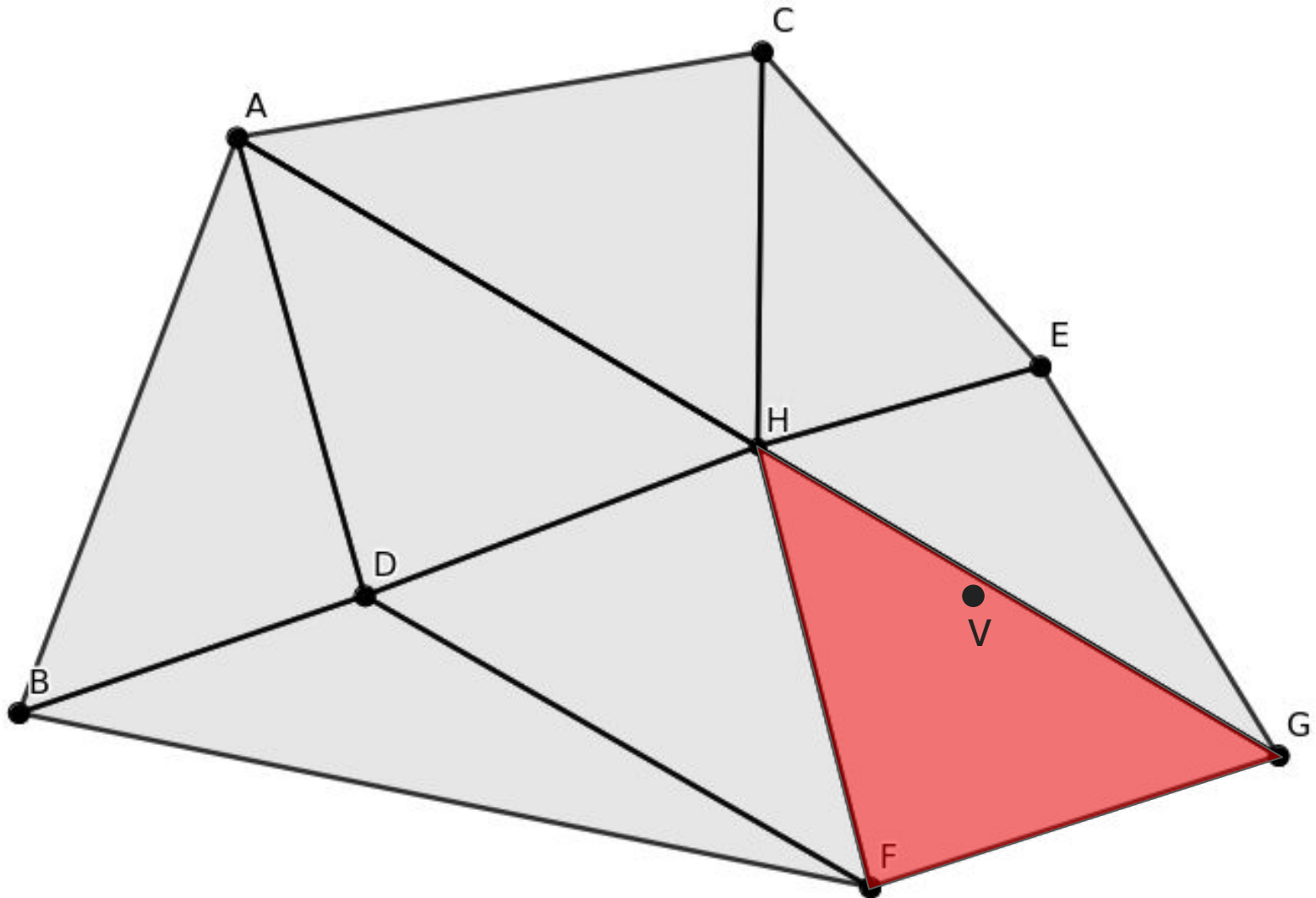
Start with a triangle of near(est) neighbor



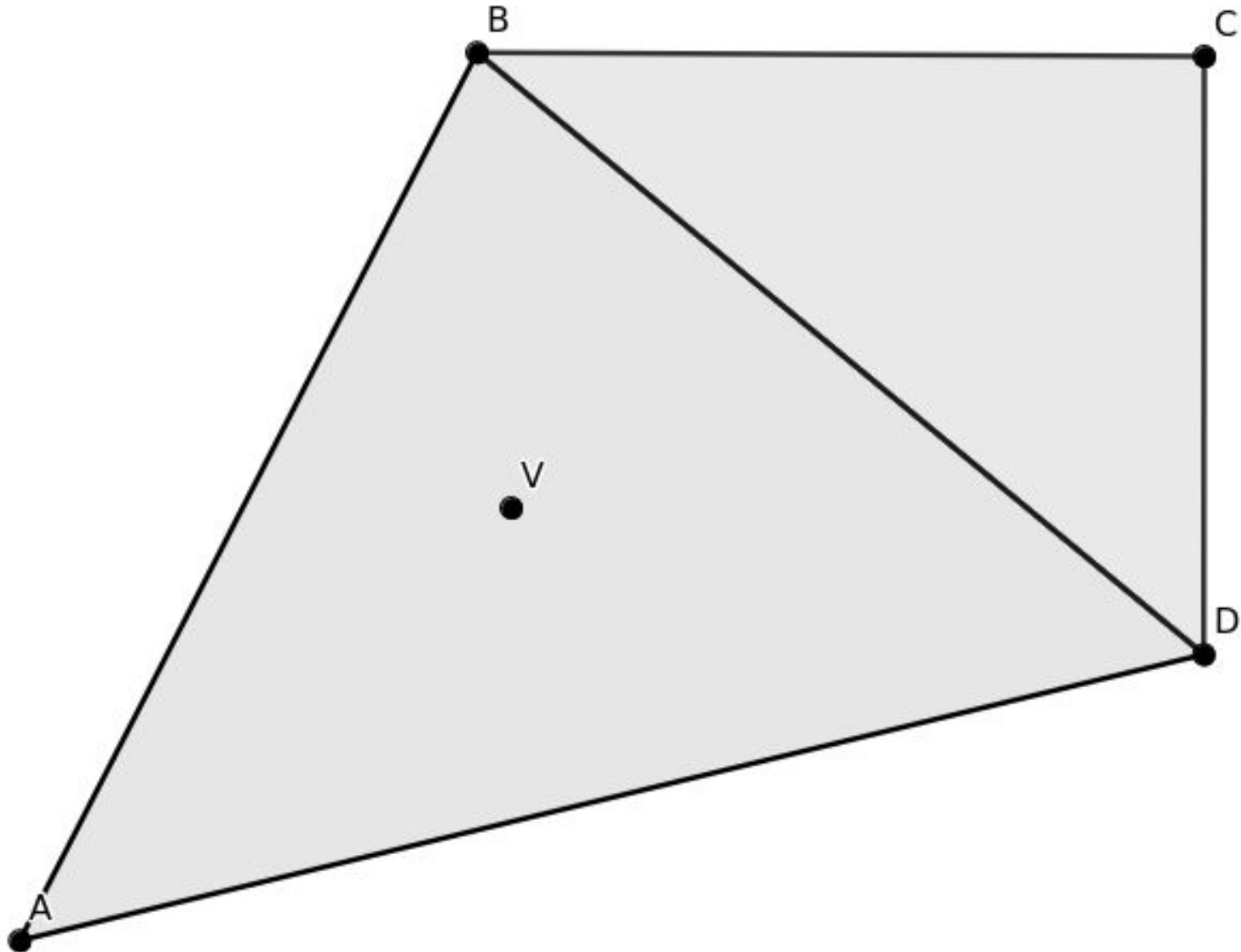
Triangle walk



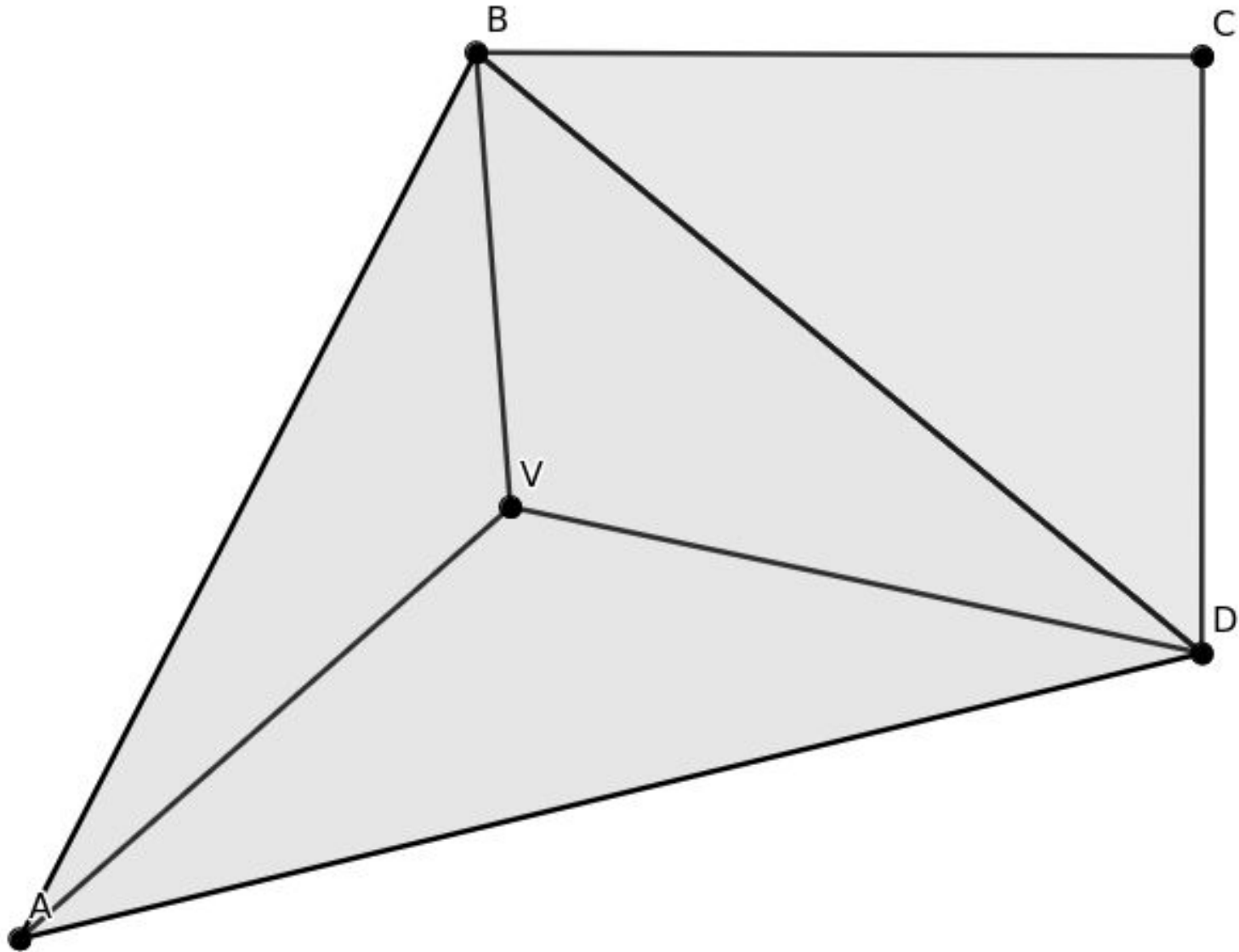
Triangle walk



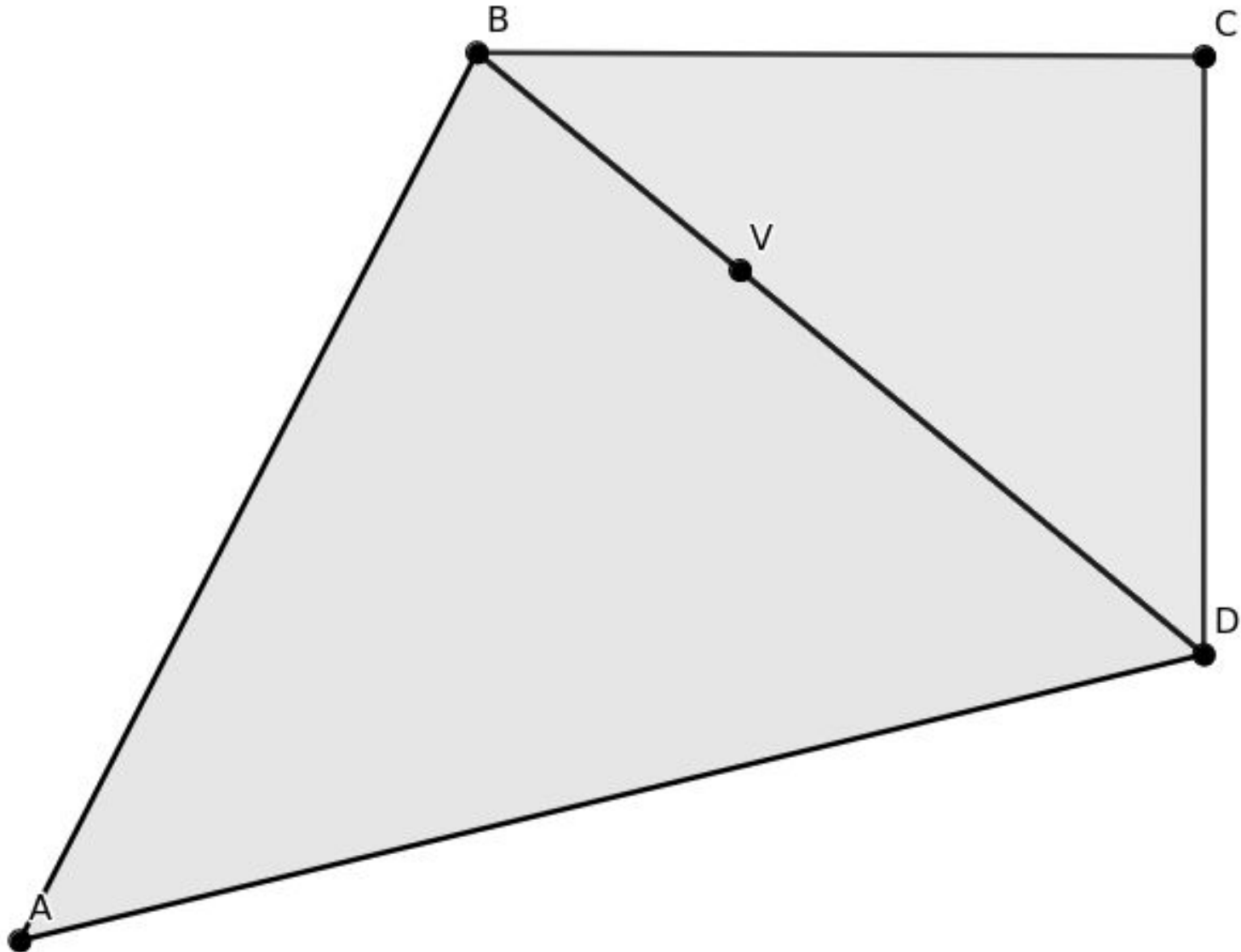
Vertex is inside a triangle



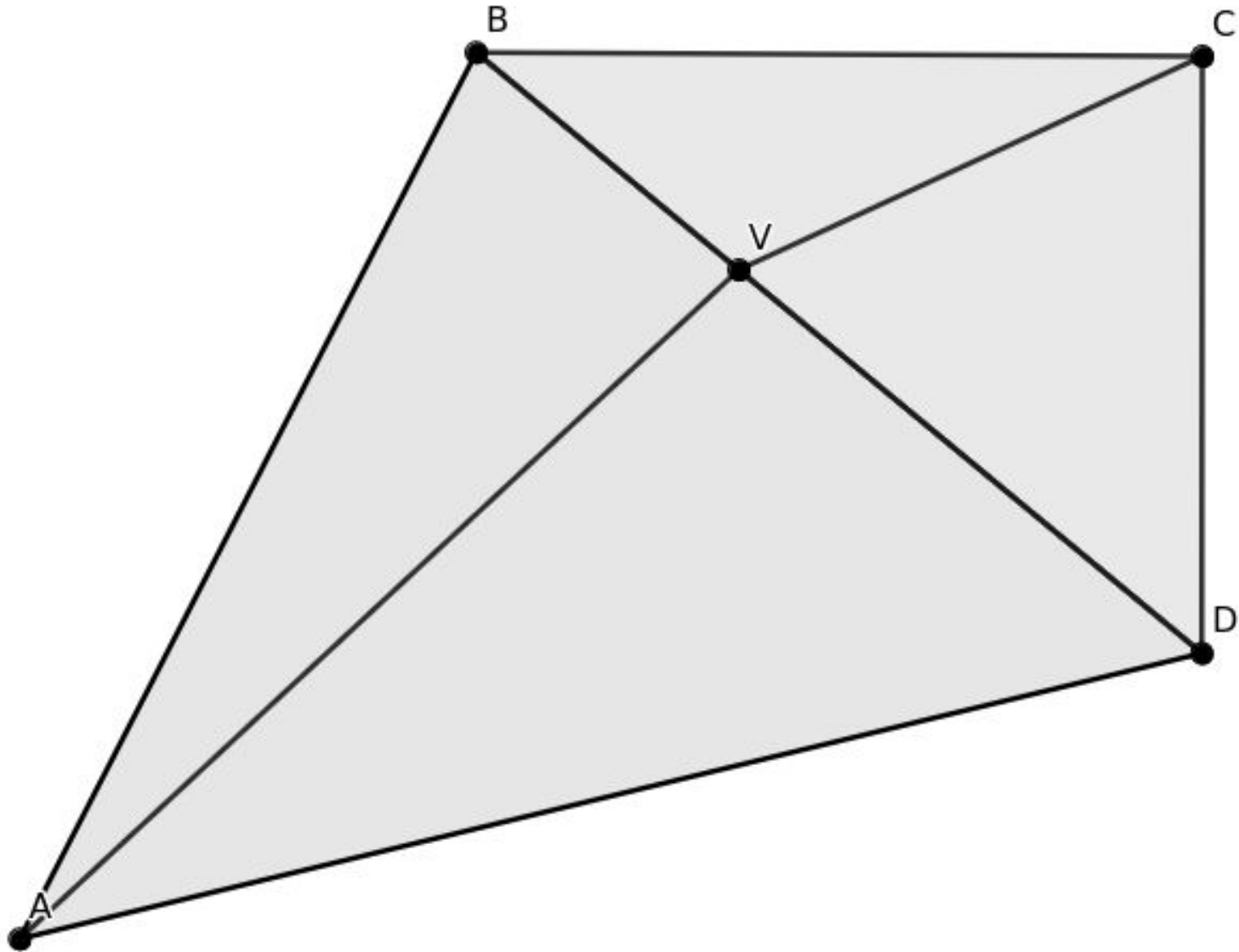
Vertex is inside a triangle



Vertex is on an edge

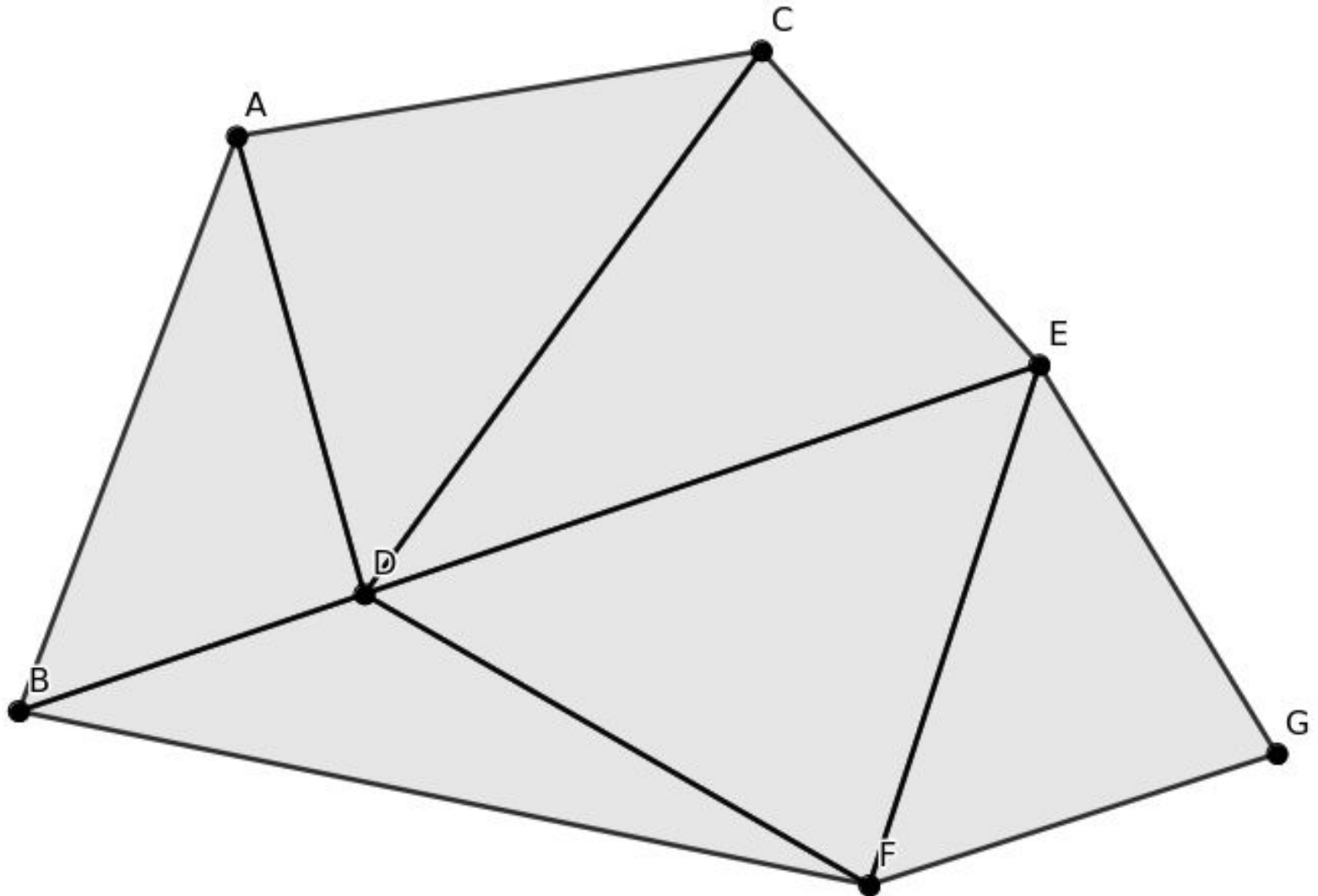


Vertex is on an edge

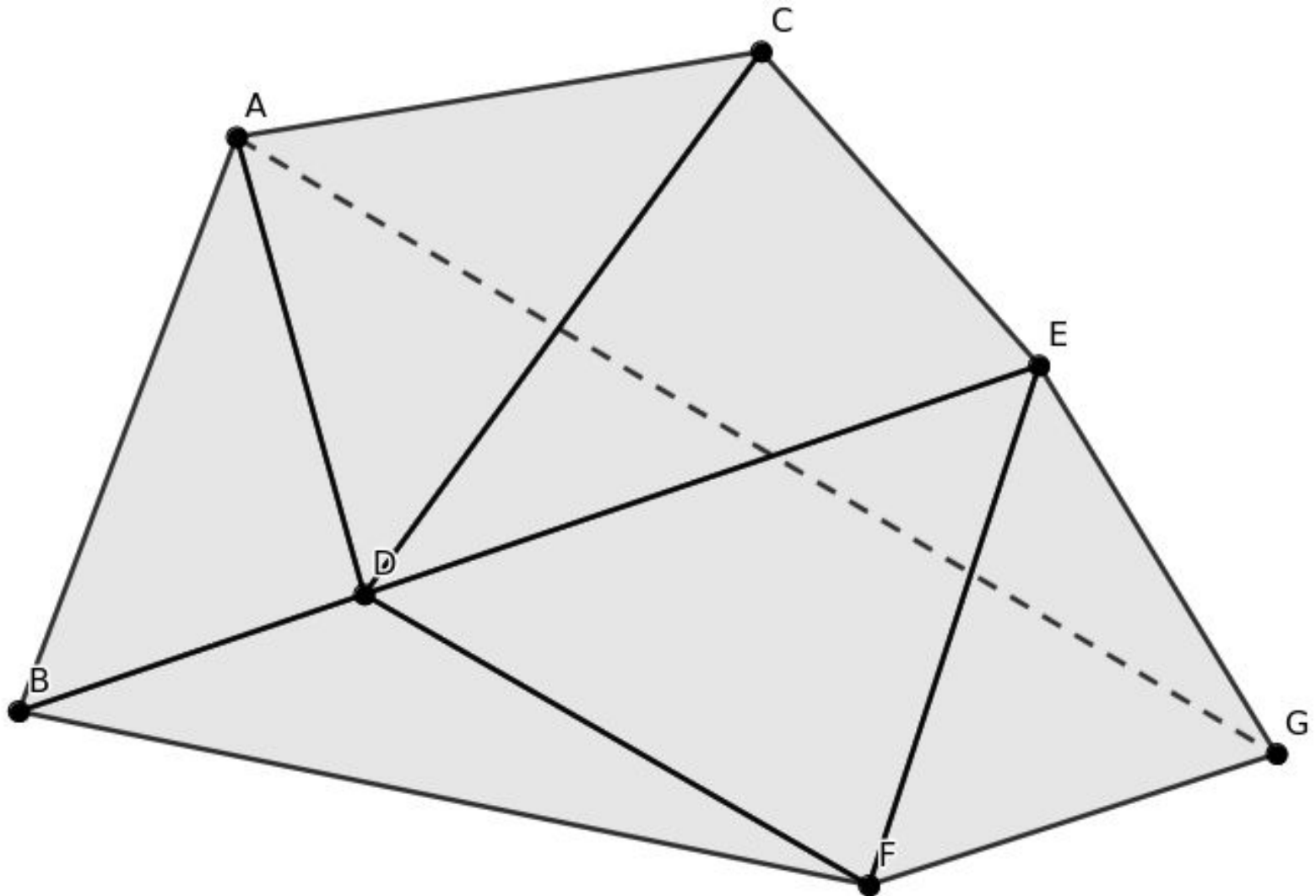


**Flip edges until all new triangles meet
Delaunay criterion**

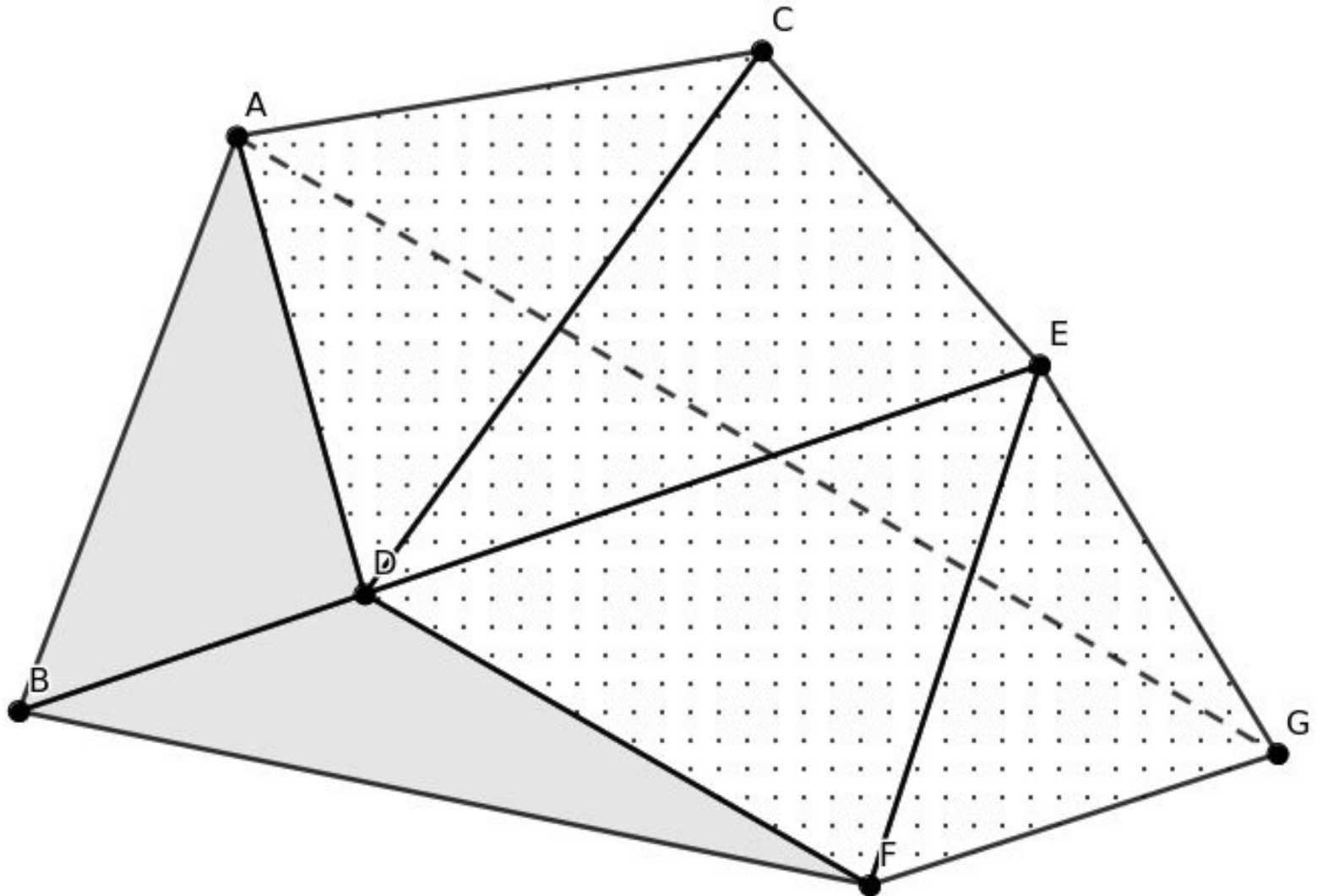
Add edge (constraint)



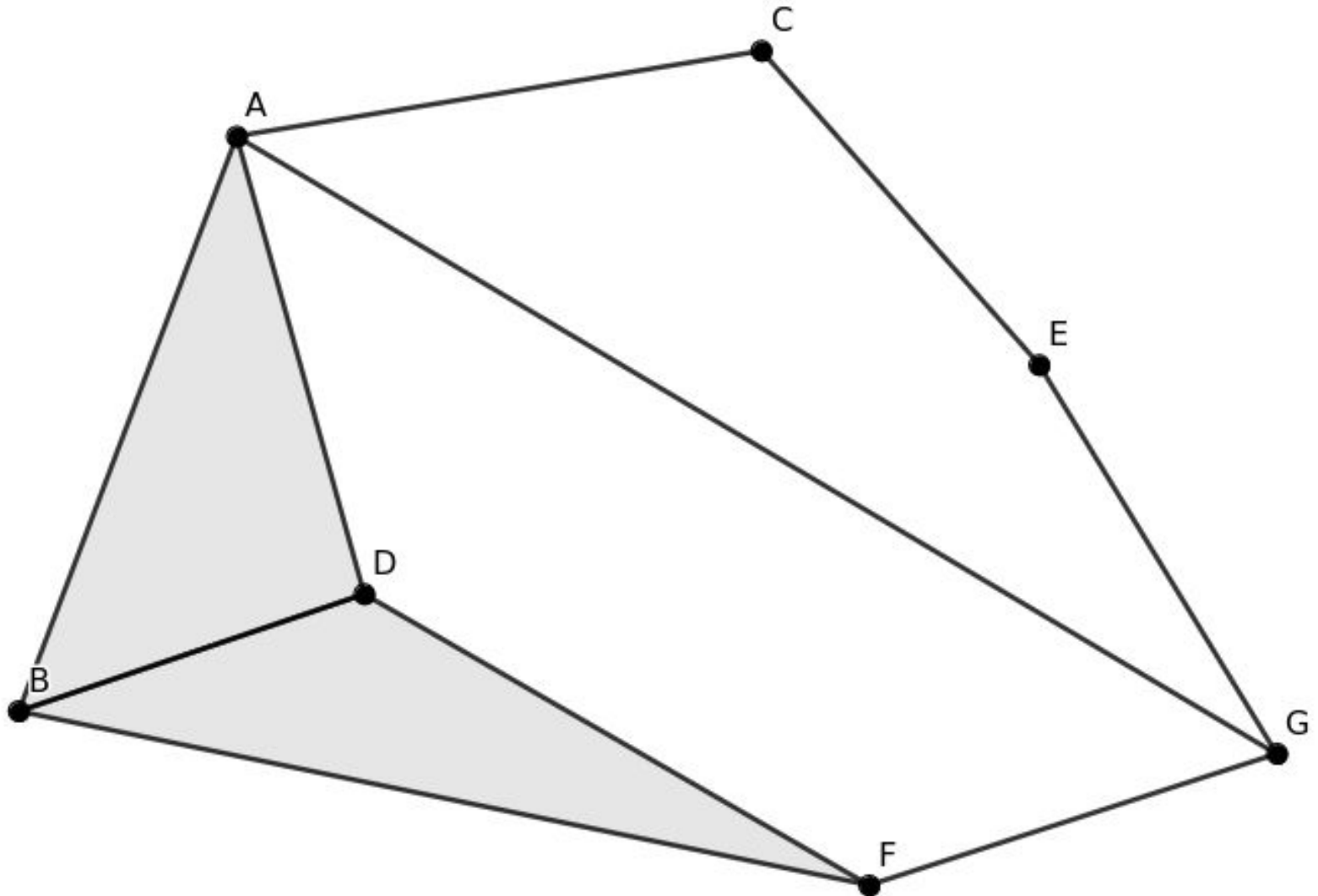
Add edge (constraint)



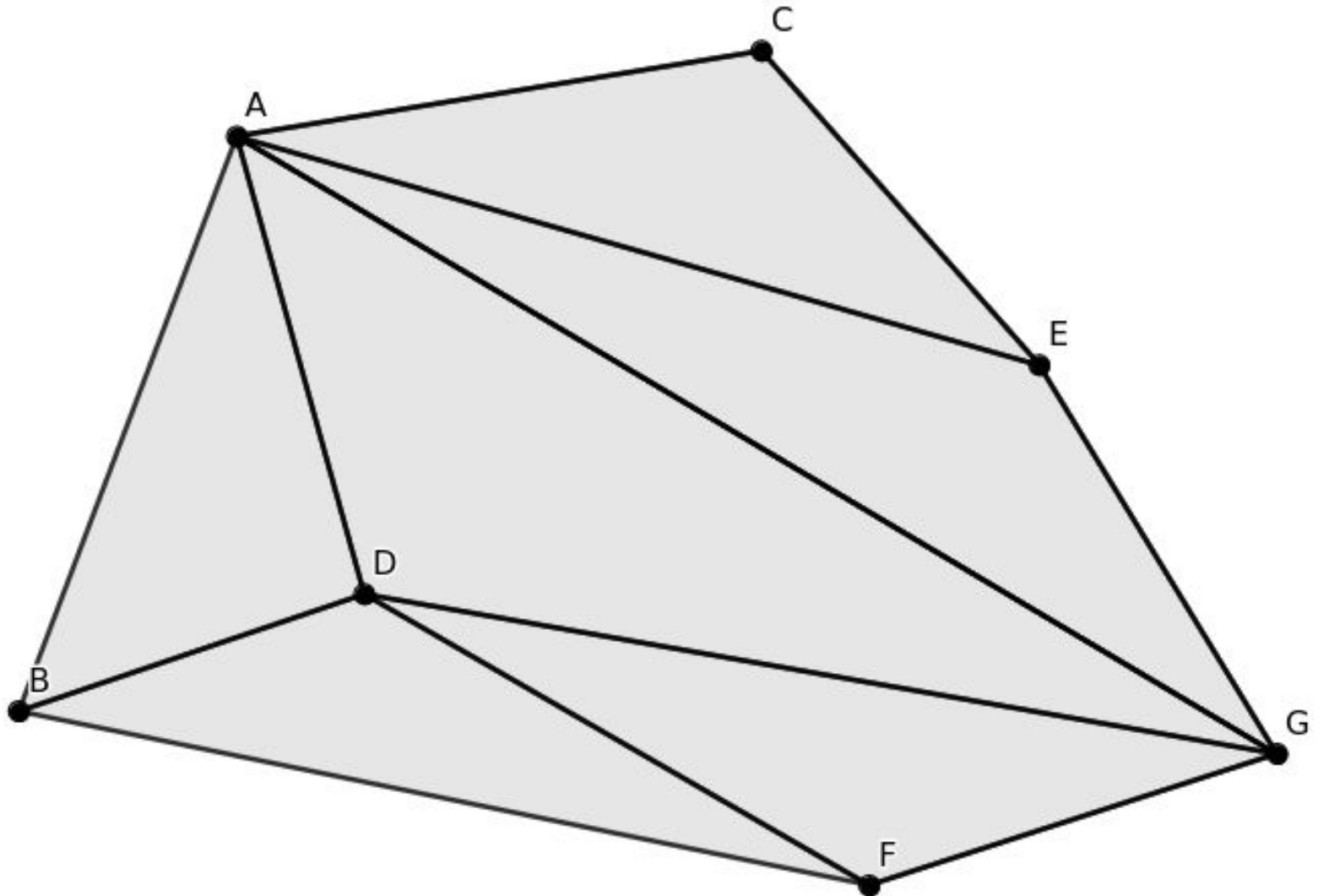
Add edge (constraint)



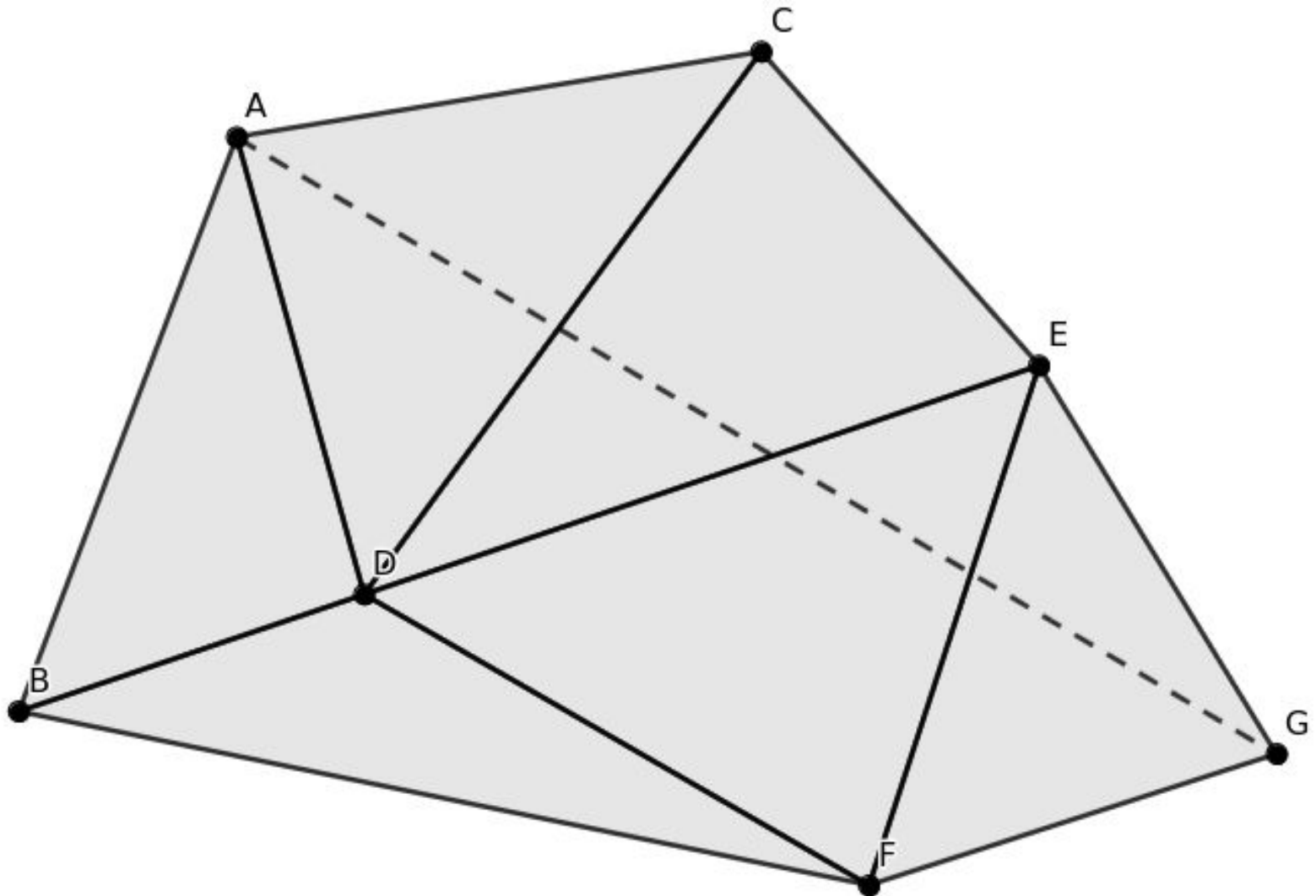
Add edge (constraint)



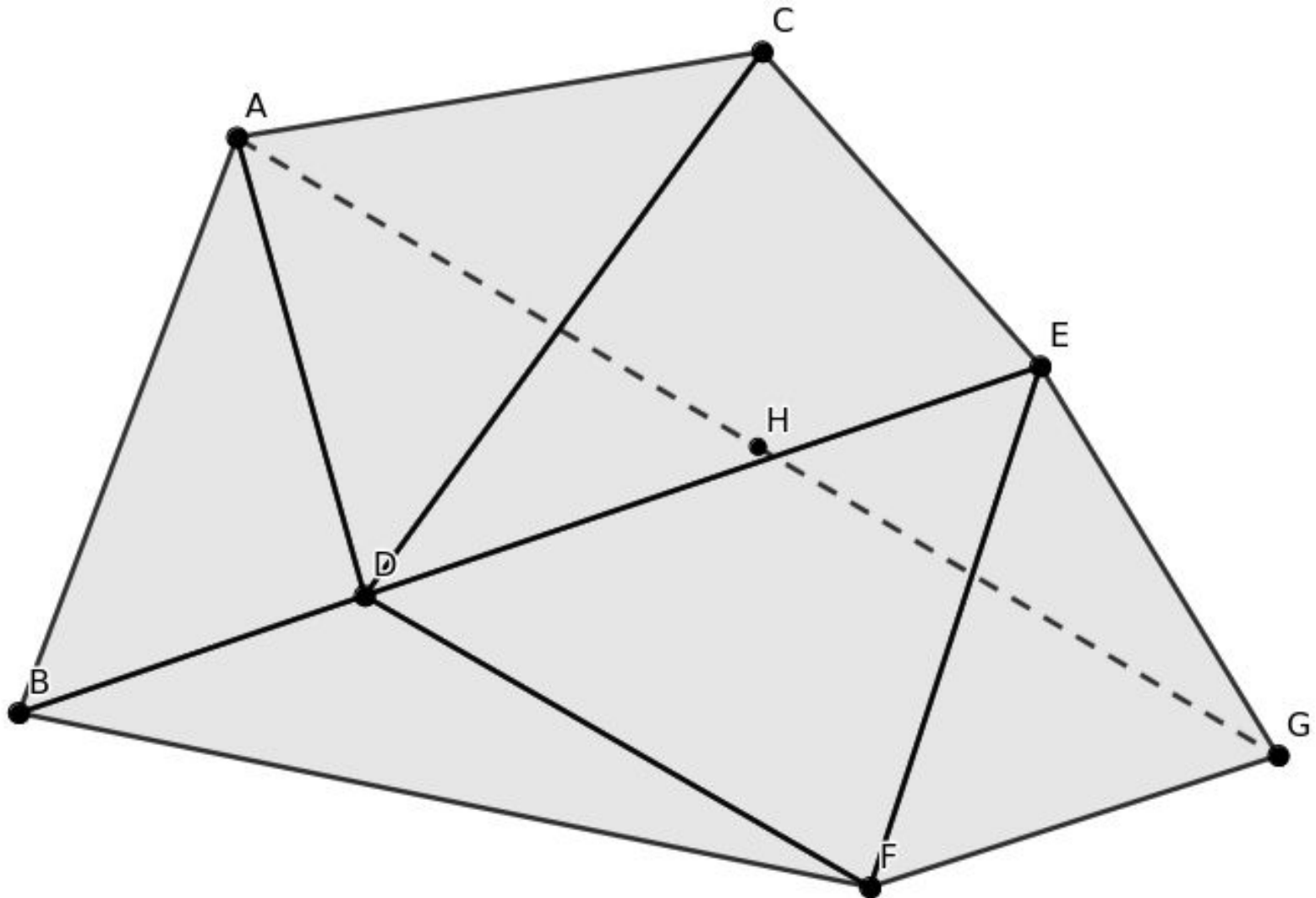
Add edge (constraint)



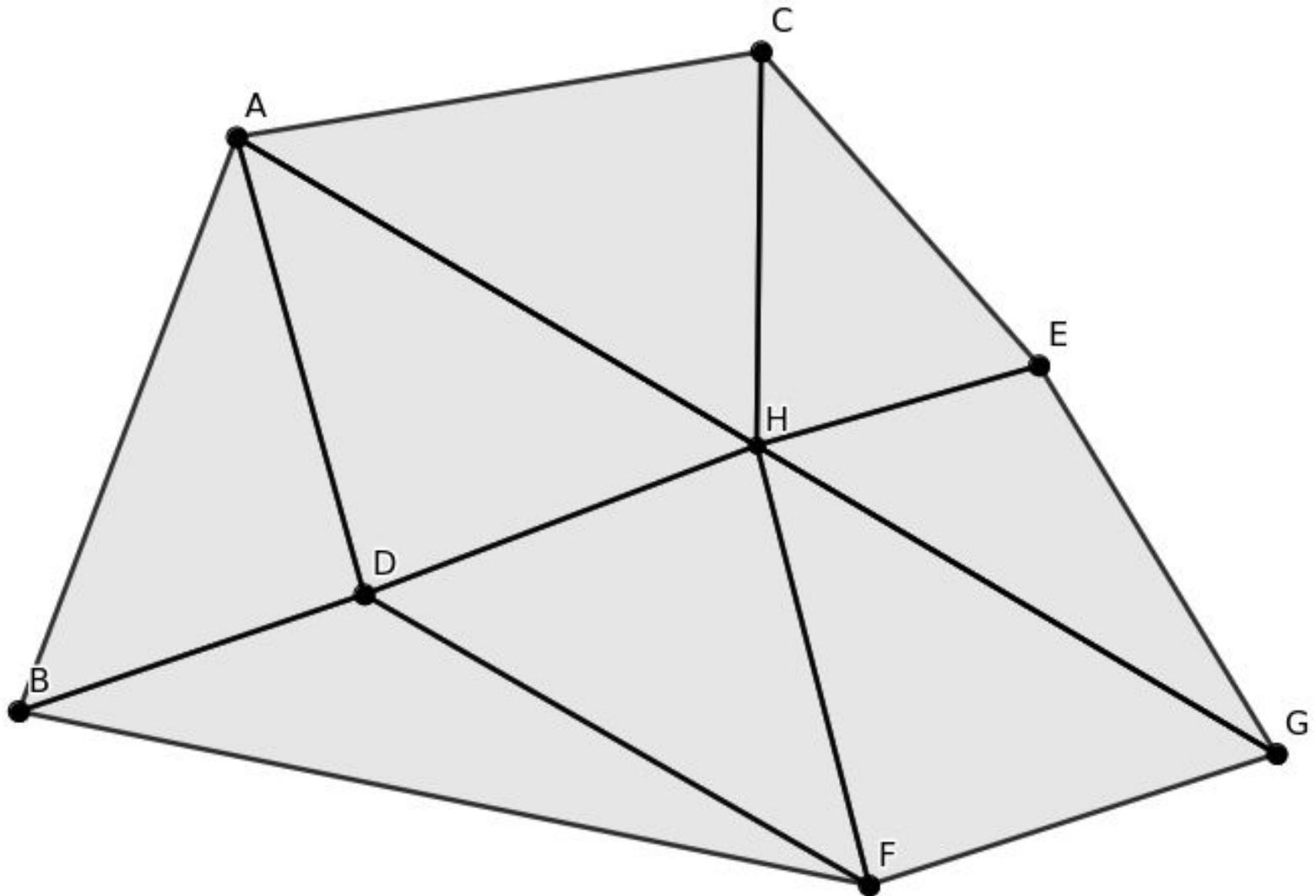
Add edge (conform)



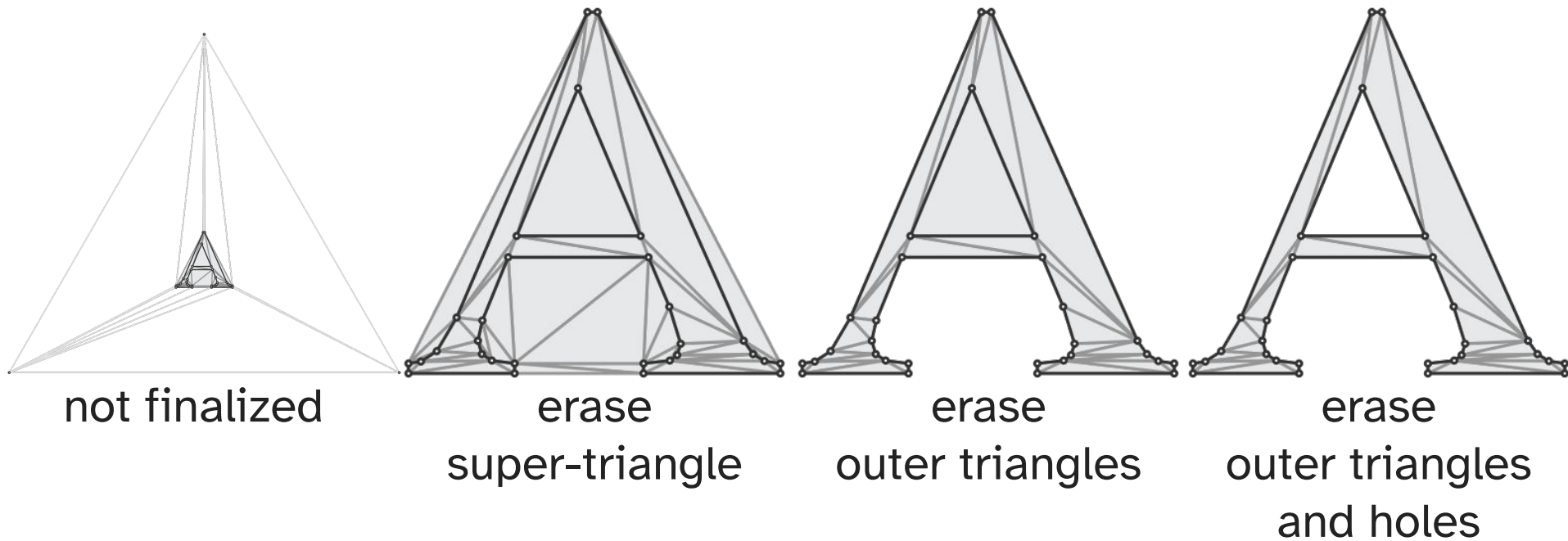
Add edge (conform)



Add edge (conform)



Finalize: remove unnecessary triangles



Core values

- Approachability
- Availability
- Compatibility
- Composability
- Debuggability
- Expressiveness
- Extensibility
- Interoperability
- Integrity
- Maintainability
- Measurability
- Operability
- Performance
- Portability
- Resiliency
- Rigor
- Robustness
- Safety
- Security
- Simplicity
- Stability
- Thoroughness
- Transparency
- Velocity

Taken from Brian Cantrill talk:

https://www.youtube.com/watch?v=Xhx970_JKX4

Core values: CDT library

- Approachability
- Availability
- Compatibility
- Composability
- Debuggability
- Expressiveness
- Extensibility
- Interoperability
- Integrity
- Maintainability
- Measurability
- Operability
- Performance
- Portability
- Resiliency
- Rigor
- Robustness
- Safety
- Security
- Simplicity
- Stability
- Thoroughness
- Transparency
- Velocity

Core values: why?

A great tool needs to be:

1. Good at doing its intended job
 - Performance, robustness, portability
2. Easy to use
 - Composability, portability
3. Can be fixed when broken and improved over time
 - Debuggability, maintainability

Core values: CDT

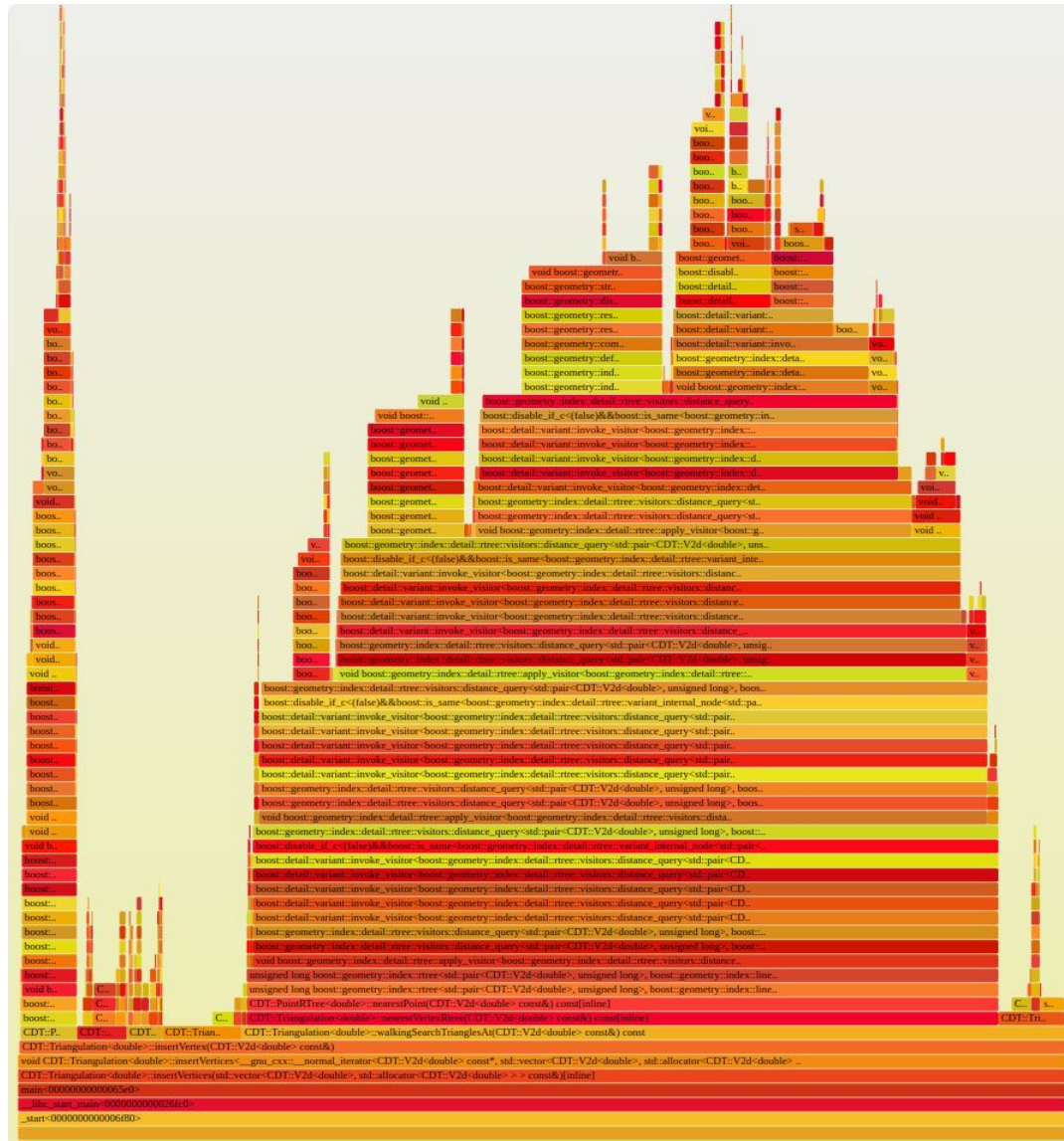
- Approachability
- Availability
- Compatibility
- Composability
- Debuggability
- Expressiveness
- Extensibility
- Interoperability
- Integrity
- Maintainability
- Measurability
- Operability
- **Performance**
- Portability
- Resiliency
- Rigor
- Robustness
- Safety
- Security
- Simplicity
- Stability
- Thoroughness
- Transparency
- Velocity

Performance improvement

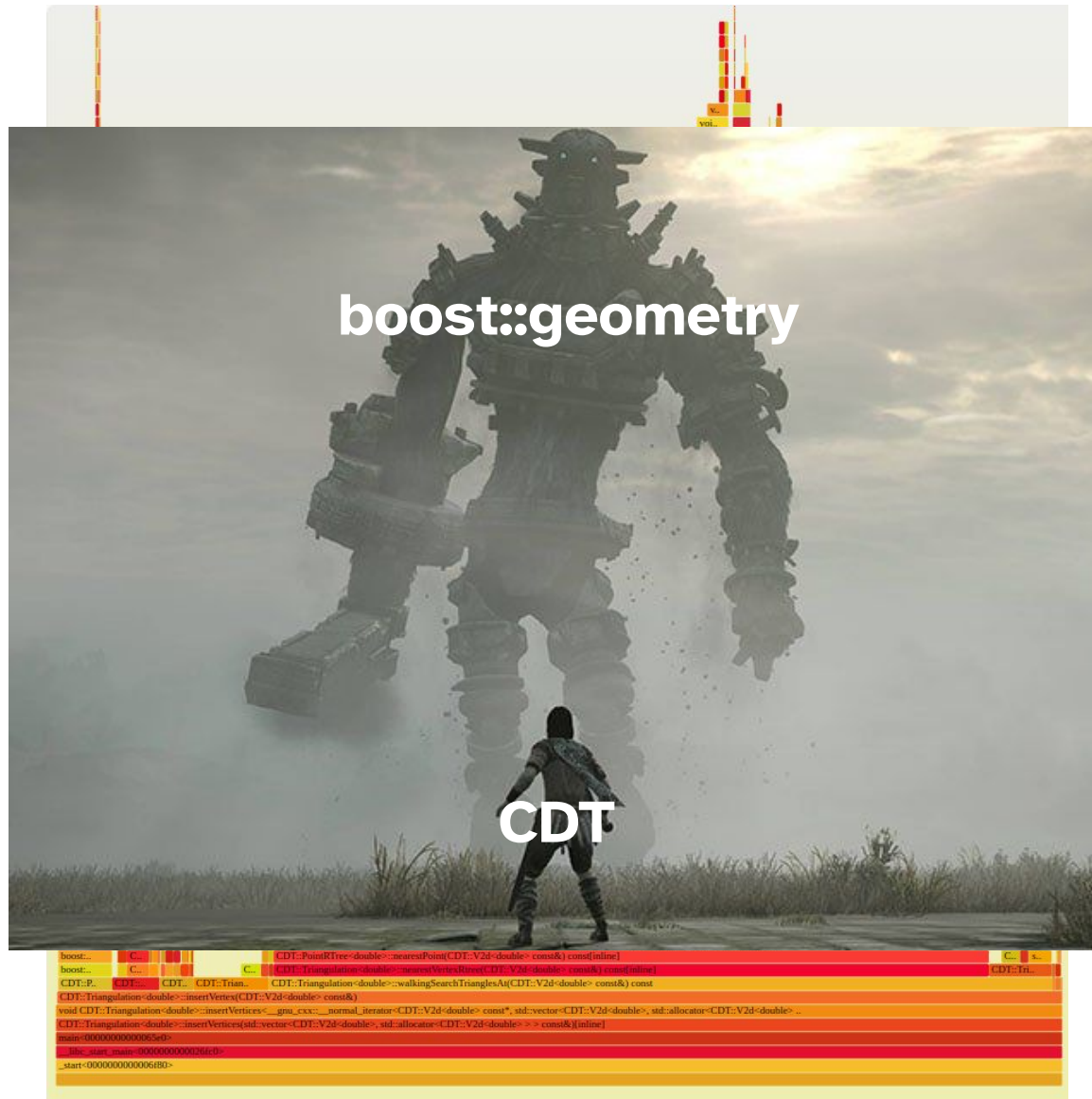
- Measuring > guesswork
- [Google perftools](#) + [Brendan Gregg's flame graphs](#) = ❤️
 - Release with debug info
 - Non-intrusive

```
LD_PRELOAD=<full_path_to>/libprofiler.so      \  
    CPUPROFILE=main.prof                      \  
    CPUPROFILE_FREQUENCY=100000               \  
    ./benchmark  
  
google-pprof --collapsed ./benchmark main.prof \  
    | grep my_benchmark                       \  
    | <full_path_to>/flamegraph.pl > flamegraph.svg
```

Flame graph 🥰



Flame graph 🥰



Flame graphs

- `boost::geometry::rtree` is the bottleneck here
- It was replaced with kd-tree

Flame graphs

- Useful
- Interactive
- Shareable

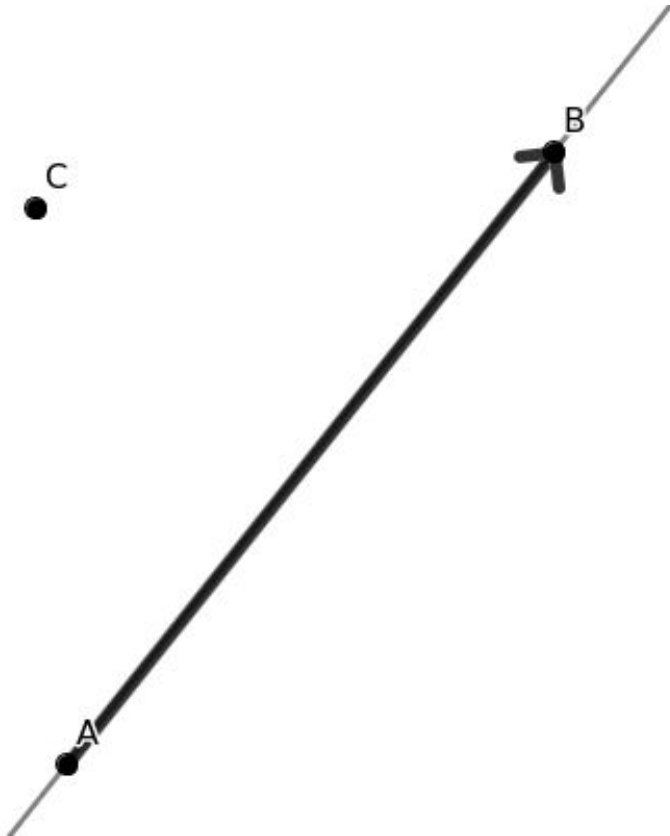
Do you use flame graphs?



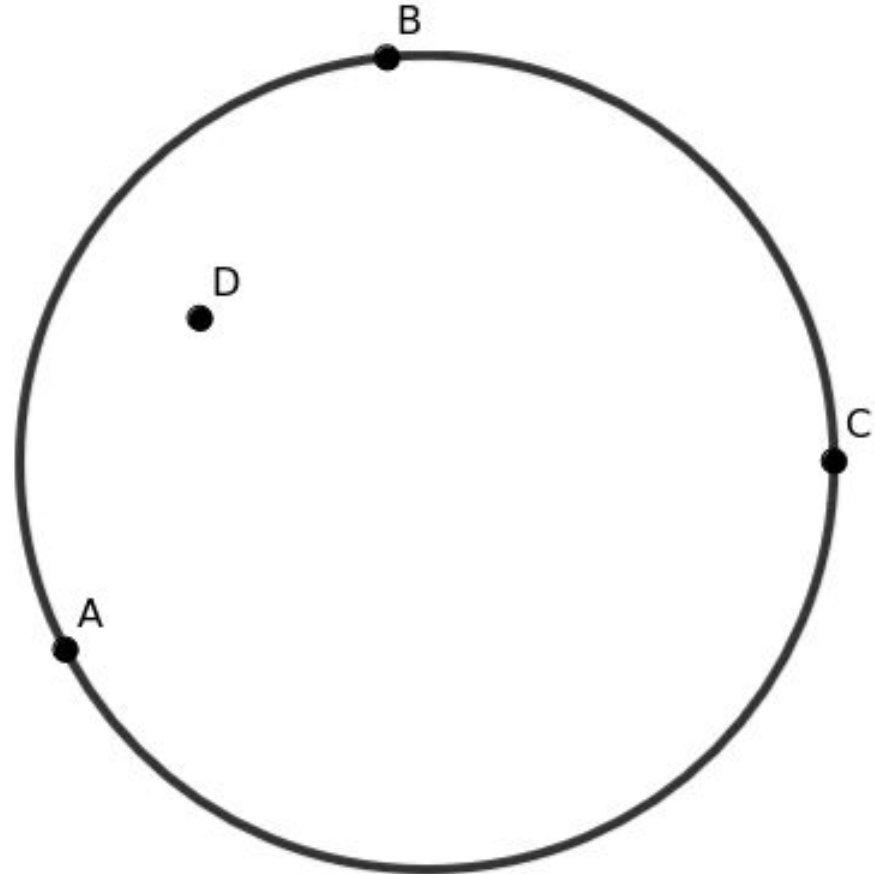
Core values: CDT

- Approachability
- Availability
- Compatibility
- Composability
- Debuggability
- Expressiveness
- Extensibility
- Interoperability
- Integrity
- Maintainability
- Measurability
- Operability
- Performance
- Portability
- Resiliency
- Rigor
- **Robustness**
- Safety
- Security
- Simplicity
- Stability
- Thoroughness
- Transparency
- Velocity

Geometric predicates



Orientation predicate



Incircle predicate

Robust (exact) geometric predicates

- Adaptive precision
 - calculate error
 - if error may affect the outcome, extend the precision
- 👍 No tolerances
- 👍 Strong guarantees of correctness
- 👎 Pedantic, often annoyingly, e.g., can produce very thin triangles

Coverage-guided fuzz testing

- Bug report from github user Some1Lse: found with fuzzing
- Found 1 extreme corner-case bug and 1 genuine bug
- In conclusion fuzzing worked great
- For details see write up by Some1Lse:

<https://gist.github.com/Som1Lse/95c2bf99385138451b614d8a94066ed7>



Fuzzing found a corner-case bug

```
// code to calculate super-triangle vertices  
// incircle radius upper bound  
auto r = std::max(box_width, box_height);  
auto super_tri_v1_y = box_center.y - r;  
// input triggering the bug  
triangulation.insertVertices({  
    {0.0, 1e38},  
    {1.0, 1e38}  
});
```


Automatically reduce the input to the minimal reproducer

- Find the smallest subset of the input triggering a problem
- Great idea, helps a lot: automates tedious manual effort 🤖
- Discovered when fuzzing
- I wish I discovered it earlier:
imagine huuuge file attached to the bug report 🤖

How it works

1. Try removing constraint edges one by one
2. Try removing vertices one by one
3. Don't forget to remap edges when removing vertices
4. Run it and go for a fika  

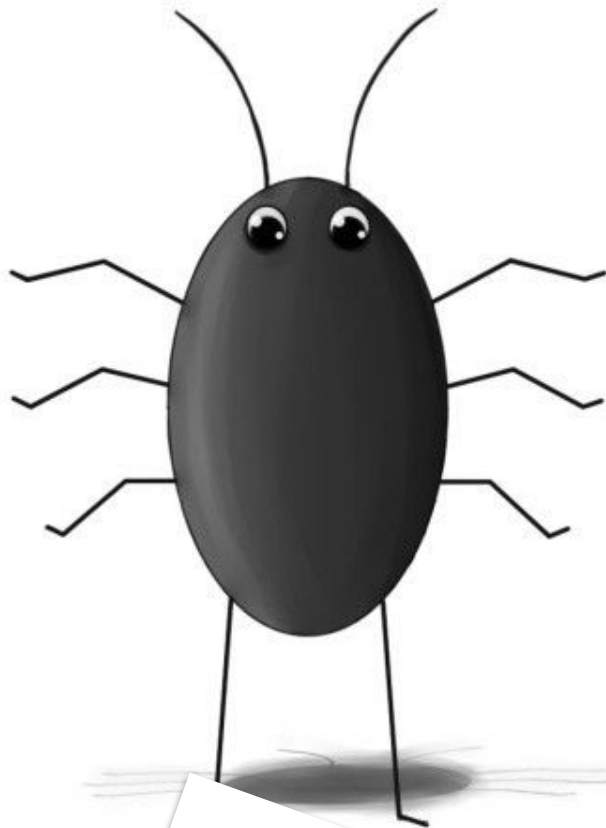
Core values: CDT

- Approachability
- Availability
- Compatibility
- **Composability**
- Debuggability
- Expressiveness
- Extensibility
- Interoperability
- Integrity
- Maintainability
- Measurability
- Operability
- Performance
- **Portability**
- Resiliency
- Rigor
- Robustness
- Safety
- Security
- Simplicity
- Stability
- Thoroughness
- Transparency
- Velocity

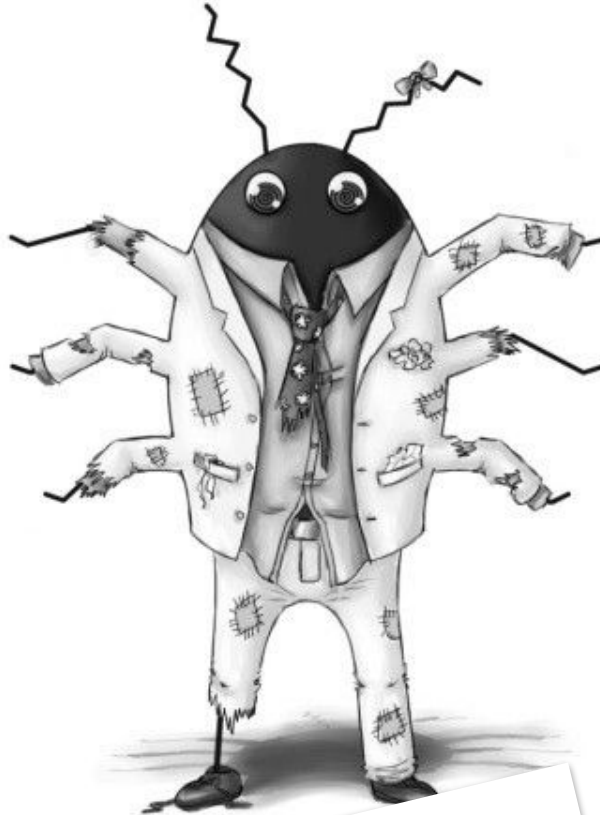
**User requests C++98 support.
What do you answer?**



Actually it is C++03 TR1, but



C++98



C++03 TR1

Backwards Compatibility with C++98 🤔

POLYFILLS, POLYFILLS EVERYWHERE

```
#ifdef CDT_CXX11_IS_SUPPORTED
...
#else
...
#endif
```

Backwards Compatibility with C++98

- 👍 Thanks heavens **boost** exists 🙏
- 👍 Appreciation for the nice things added to C++
many of which are so essential
- 👎 Maintaining backwards compatibility is hard
(latest version is not compatible because of `std::to_string`)
- 👎 Makes maintenance and accepting contributions harder

Minimal dependencies

- Only dependency is robust predicates by William Lenthe
 - single header bundled in the code
 - github.com/wlenthe/GeometricPredicates
- boost is needed for C++98 polyfills
- Minimizing dependencies is a trade-off
 - 👍 Improves portability
 - 👍 Makes library easy to use (also licensing wise)
 - 🙅 Forces to reinvent the wheel (*great artists steal*)
 - 🙅 Easier to achieve for a small and focused library

Integrating with user types

```
struct MyPoint2D { double coord[2]; };
triangulation.insertVertices(
    points.begin(),
    points.end(),
    [](const MyPoint2D& p){ return p.coord[0]; },
    [](const MyPoint2D& p){ return p.coord[1]; }
);

struct MyEdge { std::pair<size_t, size_t> verts; };
triangulation.insertEdges(
    edges.begin(),
    edges.end(),
    [](const MyEdge& e){ return e.verts.first; },
    [](const MyEdge& e){ return e.verts.second; }
);
```

Features that make CDT easier to use

- PythonCDT: python bindings
- Package managers: vcpkg, spack, Conan
- Header only or compiled

Header-only Vs. compiled
Do you have a strong preference?



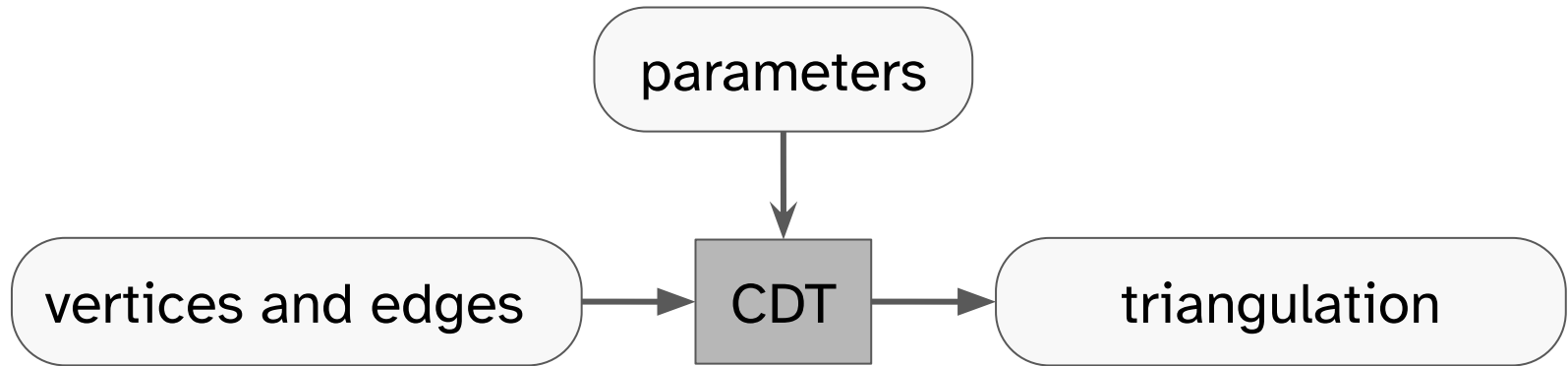
Core values: CDT

- Approachability
- Availability
- Compatibility
- Composability
- **Debuggability**
- Expressiveness
- Extensibility
- Interoperability
- Integrity
- **Maintainability**
- Measurability
- Operability
- Performance
- Portability
- Resiliency
- Rigor
- Robustness
- Safety
- Security
- Simplicity
- Stability
- Thoroughness
- Transparency
- Velocity

Use indices instead of iterators

- Used to identify vertices and triangles
- 👍 No iterator invalidation, e.g., when growing the vector
- 👍 Same index can be applied to multiple vectors
- 👍 Index can be made smaller (32bit)
- 👍 Indices can be easily used as labels in visualizations

Data-driven tests



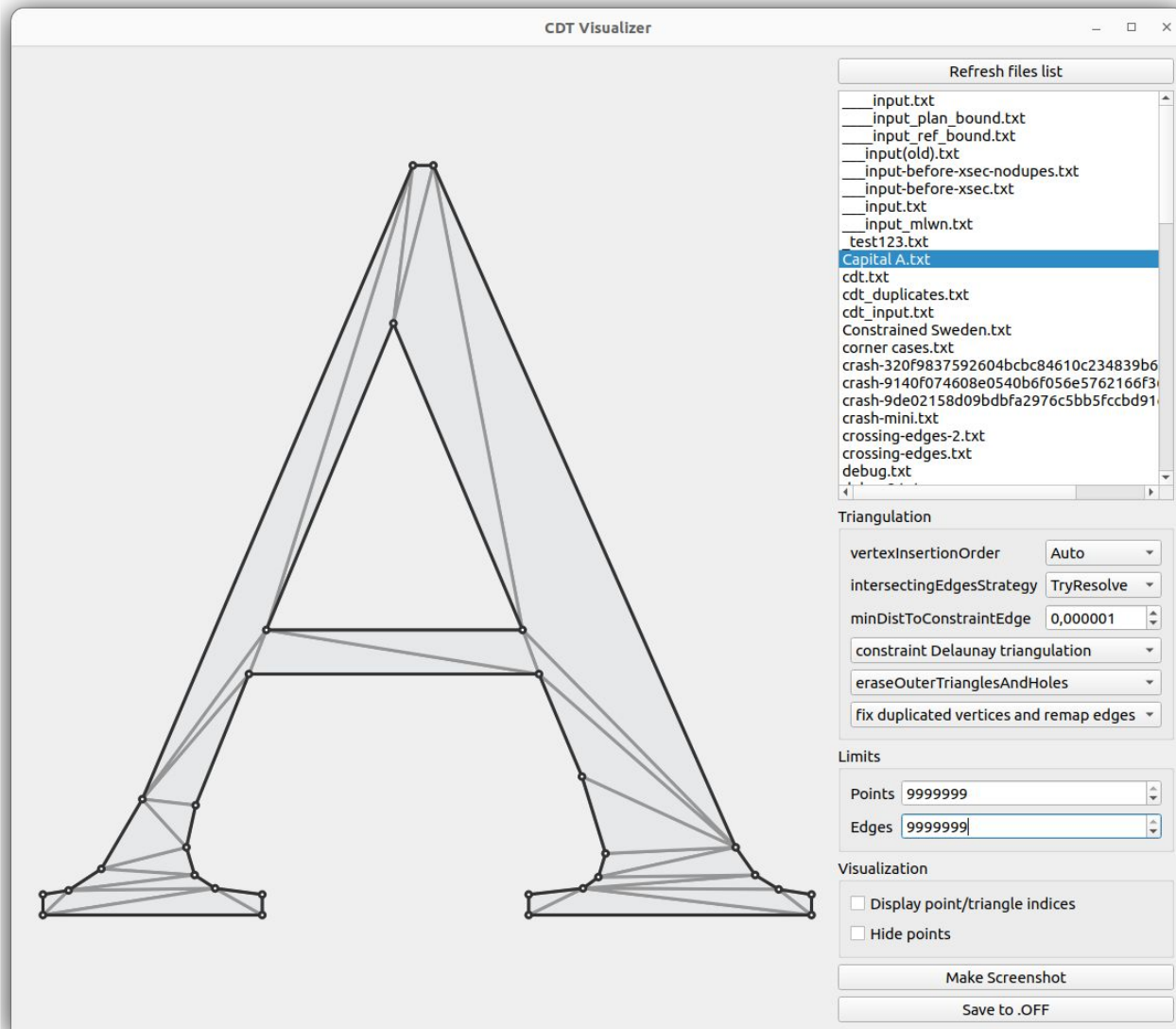
Data-driven tests

- Text formats for storing the input and output
 - Diffable
 - Comparable: as string or via a checksum
- Tests inputs and expected outputs are stored as files

Error handling

- Asserts are used heavily-ish
- Exceptions that help to pinpoint the problem
 - *“Intersecting edges: (1,2) intersects (3,4)”*
 - *“Duplicate vertices: 1 is a duplicate of 2”*

Visualizer tool demo



Online prototype



Determinism (reproducibility)

A 3D rendering of a ping pong table with a green top and white lines. Two red paddles with wooden handles are positioned on opposite sides of the table. A white ping pong ball is in the center of the table. The text "Game with physics example" is overlaid in the center.

Game with physics example

Determinism (reproducibility)

- Program is **deterministic** if running on the same inputs will always produce the same outputs
- Reproducibility makes debugging a looooooot easier
Bugs are easier to find and fix when they can be reliably reproduced

Is `std::nth_element` deterministic?

Is `std::nth_element` deterministic?

```
std::vector<std::array<int, 2> > v =  
{  
    {1, 42}, {2, 42}, {3, 42}, {4, 42}  
};  
std::nth_element(  
    v.begin(),  
    v.begin() + v.size() / 2,  
    v.end(),  
    [](const auto& lhs, const auto& rhs)  
    {  
        return lhs[1] < rhs[1];  
    })  
);
```

Is `std::nth_element` deterministic?

libstdc++

The median is {1, 42}

v is {{3, 42}, {4, 42}, {1, 42}, {2, 42}}

libc++ and msvc stl

The median is {3, 42}

v is {{1, 42}, {2, 42}, {3, 42}, {4, 42}}

Is `std::nth_element` deterministic?

- Solution: include implementation from **libc++**
- Can we have `std::stable_nth_element`?

Fast and deterministic pseudo-random number generation (PRNG)

- Original implementation used **mt19937** which is slow and heavy
- Switched to **SplitMix64**
 - Small: 8 bytes of state vs. 2504 bytes of mt19937
 - Fast: approx twice as fast as mt19937
 - Poor for cryptographic purposes
 - Good enough for our purposes
 - Unfortunately not in standard library 🙄
- Fixed the seed of PRNG inside CDT API for determinism

SplitMix64 code fits on a single slide

```
/// SplitMix64 pseudo-random number generator
struct SplitMix64
{
    uint64 state;
    explicit SplitMix64(uint64 state) : state(state) {}
    explicit SplitMix64() : state(0) {}
    uint64 operator()()
    {
        uint64 z = (state += 0x9e3779b97f4a7c15);
        z = (z ^ (z >> 30)) * 0xbf58476d1ce4e5b9;
        z = (z ^ (z >> 27)) * 0x94d049bb133111eb;
        return z ^ (z >> 31);
    }
};
```

Promoting

- Asking for stars in the README
- Github SEO: optimized about section
 - Updates instantaneously
 - <https://www.markepear.com/blog/github-search-engine-optimization>
- Links on Wikipedia

Licensing

MPL-2.0 is neat: permissive, yet weak copyleft

Community

Thank you, thank you, thank you!

Contributors:

[Karl Åkerblom](#), [baiwenlei](#), [Bärbel Holm](#), [Andre Fecteau](#), [msokalski](#),
[alkonior](#), [ldcMasa](#), [eqladil86](#), [Som1Lse](#), [zhivkob](#), [here-abarany](#),
[Islam0mar](#), [icortial-safran](#), [pageldev](#), and others

- Finding bugs, fixing bugs, suggesting the fixes
- Great discussions, ideas, suggestions
- Benchmarking, profiling

Summary and lessons

- What makes library successful
 - solve important problem that many people have
 - easy to use
 - right place at the right time, fill an empty niche
- Community is a super-power
- Determinism greatly improves debuggability
- Make most of the use of the fact that the problem is visual
- Invest in tooling to make debugging/maintaining easy and fun

Thank you!

I'm happy to connect, e.g., on linkedin or github

Please fill in 2 min poll at
<https://forms.gle/gf9copYzuqDfXs3GA>

