# 二维 (伪) 拓扑图生成器

余政希 [1]

[1] 同济大学 国豪书院

## Overview

本项目实现了一个跨平台的二维（伪）拓扑图生成工具，支持通过参数化规则快速生成具有不同层次、不同连接、不同纹理特征的网状结构图案。本项目提供实时渲染与交互浏览（缩放、平移、节点拖拽），并支持一键导出为 PNG/SVG，用于算法可视化、图形设计草图与教学演示等。本项目基于 C++17 与 Qt6，并将“生成算法”和“可视化交互”打包为不同模块，便于扩展更多生成模式与样式预设。

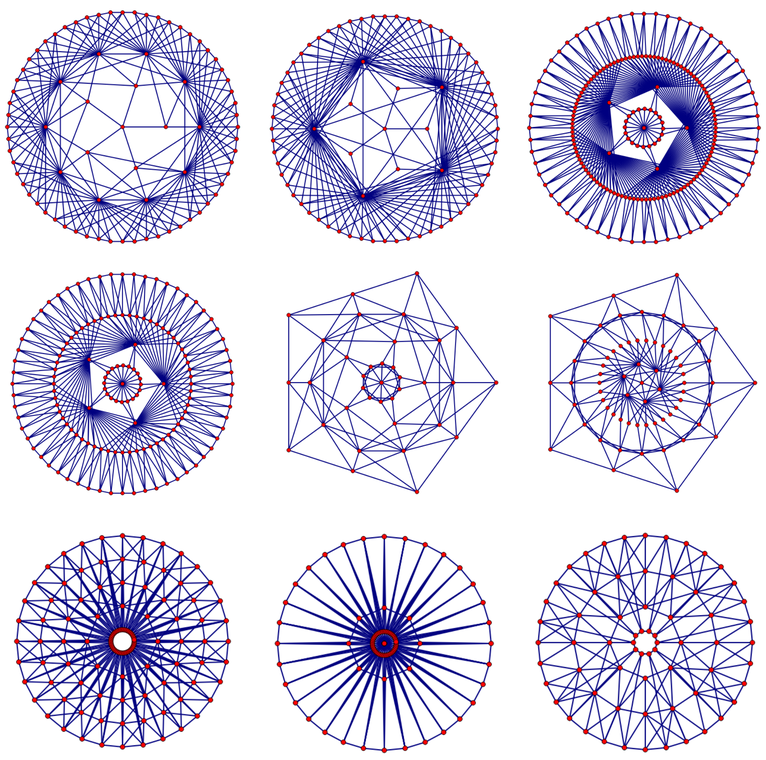

Figure 1. Layered Polygon Mode（第一、二排），Concentric Mode（第三排）

## Highlights

- **双生成范式：**包含 Concentric Mode（同心环网状）与 Layered Polygon Mode（分层多边形/星形结构）两种不同生成范式。
- **规则可组合：**支持层内封圈、对角/星形、层间 $1\to1$ 最近、$1\to2$ 扩展、左右交替 Zigzag 等自定义，可通过参数自由组合形成丰富图案。
- **交互即数据：**节点可拖拽，边实时跟随；用于“生成后再编辑”的交互式探索流程。
- **所见即所得：** PNG 位图与 SVG 矢量一键导出。
- **模块易扩展：** Generator 仅负责 $V, E$，Scene/View 负责图元与交互，便于新增扩展模块。

## Quickstart

**本项目依赖以下环境**

1. Cmake >= 3.21
2. C++17 编译器（clang++ / g++ / MSVC）
3. Qtt6

macOS（Homebrew）

```
brew install cmake qt@6
cmake -S . -B build -DCMAKE_BUILD_TYPE=Release
cmake --build build -j 8
./build/src/topology_diagram_generator
```

Ubuntu / Debian

```
sudo apt update
sudo apt install -y build-essential cmake \
    qt6-base-dev qt6-base-dev-tools
cmake -S . -B build -DCMAKE_BUILD_TYPE=Release
cmake --build build -j 8
./build/src/topology_diagram_generator
```

Windows（VS 2022）

推荐使用 Visual Studio 2022，确保已安装并正确配置：

1. Visual Studio 2022（包含“Desktop development with C++”工作负载）
2. Qt 6.x for MSVC 2022 64-bit
3. Qt VS Tools
4. CMake 3.21+，通常在安装 Visual Studio 2022 时包含。

打开 Visual Studio 2022 后，选择“打开文件夹（Open Folder）”并指定本项目根目录。

若 Qt VS Tools 插件配置正确，Visual Studio 将自动检测并加载项目的 CMake 配置文件。此时只需点击上方的“生成并运行”（Build and Run）按钮，即可直接编译并启动项目。

本项目已在 Windows 11 24H2（x64）+ Visual Studio 2022 + Qt 6.10.0 + Qt Visual Studio Tools、macOS（Qt 6.9.3 + Visual Studio Code + CMake 4.1.2）以及 Ubuntu（Qt 6.6.3）等环境下成功编译通过。

## System Structure

系统采用四层架构设计，确保核心算法与界面显示完全解耦：

1. **UI 控制层** (MainWindow)：监听参数变化，组装 Params 结构体，驱动视图更新。
2. **视图管理层** (TopologyView / Scene)：作为“画布管理器”。View 负责视口交互（缩放/平移）；Scene 负责图元的生命周期管理（创建、清空、索引）。
3. **图元表现层** (NodeItem / EdgeItem)：基于 Qt Graphics View 框架，实现具体的绘制逻辑与鼠标事件响应。
4. **核心算法层** (TopologyGenerator)：执行几何计算，输出无状态的 std::vector<Node> 和 Edge 数据，不依赖任何 UI 组件。

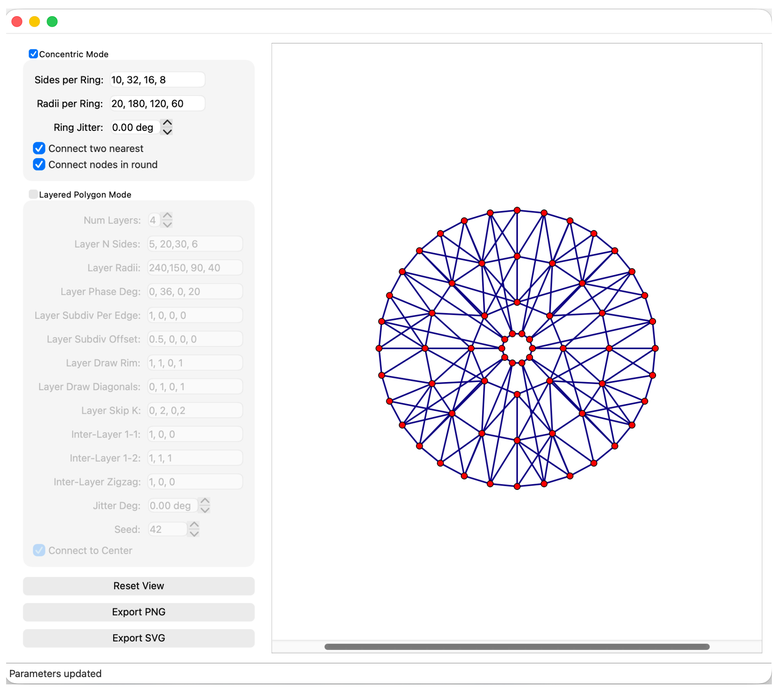


Figure 2. 程序运行主界面

## Algorithm - Concentric Mode

用若干条同心“环”快速生成网状结构。每一环在圆周上进行等角采样，并对角度加入少量随机抖动以产生更自然的艺术效果；可选择是否将每一环封成多边形“环边”，以及是否将外层点连接到内层的两个最近点，从而形成稠密的网状连接。

**输入参数**

- Radii per Ring：各环半径序列（从外到内或内到外均可，按输入顺序生成）。
- Sides per Ring：各环点数（即第 $i$ 环为“$n_i$ 边形”对应的 $n_i$）。
- Ring Jitter：每个点角度抖动（单位：度），用于增强随机性/艺术化。
- Connect nodes in round：是否绘制每一环的“环边”（首尾相连）。
- Connect two nearest：是否将每个外环点连接到内环的两个最近点。

**生成方式**对第 $i$ 个环：

1. 取半径 $r = r[i]$，点数 $n = n_{\text{sides}}[i]$。
2. 第 $j$ 个点（$0 \le j < n$）角度为
$$\theta_{i,j} = \frac{2\pi j}{n} + U(-\delta, +\delta),$$
其中 $\delta$ 为由输入抖动角度（度）换算得到的弧度。
3. 将极坐标转为直角坐标并压入节点表，同时把该环的节点 id 记录到 `rings[i]`。

说明：这里的“多边形”本质上是对圆周做等角采样形成的折线近似。

**复杂度分析**

- 点生成：$\mathcal{O}(\sum_i n_i)$
- 层内环边：$\mathcal{O}(\sum_i n_i)$
- 层间连接（相邻两环）：每对相邻环为 $\mathcal{O}(n_i \cdot n_{i+1})$

## Challenges & Solutions

- **难点 A: 非均匀采样下的层间连接跳变**
解决方案: 引入局部邻域搜索。
对每层节点计算 `atan2` 极角序列；层间匹配时，先通过角度寻找“主最近点”，再基于索引在其左右邻域进行 1-2 扩展，避免了全局搜索的不稳定性。
- **难点 B: 高频参数调节导致的渲染卡顿**
解决方案: 采用逻辑渲染分离 + 批量更新策略。
`Generator` 仅输出纯 C++ struct (Node/Edge)；`Scene` 接收数据后，先 `clear()` 旧图元，再批量 `addItem()`，最后仅触发一次 `update()`，杜绝碎片化重绘。
- **难点 C: 导出图片与屏幕显示坐标不一致**
解决方案: 统一坐标系。
使用 `mapToScene(viewport()->rect()).boundingRect()` 获取当前视口的精确场景坐标，将其作为 `QImage` 和 `QSvgGenerator` 的渲染源区域 (`sourceRect`)。

## Algorithm - Layered Polygon Mode

按“外 $\to$ 内”绘制多层 $n_l$ 边形；每条边可细分 $0/1/2$ 个额外点（支持对称偏移）。层内可选绘制环边与星形/对角线；层间连接支持最近 $1\to1$ 与扩展 $1\to2$ （并可按奇偶交替左右形成“之”字形）。中心点可设置为必连最内层。

**输入参数**

- Num Layers：层数（建议 3–4）。
- Layer N Sides：第 $l$ 层多边形边数 $n_l \ge 3$。
- Layer Radii：第 $l$ 层半径 $R_l$。
- Layer Phase Deg：第 $l$ 层起始角（度），用于层间错位旋转（如五边形相差 $36°$）。
- Layer Subdiv Per Edge $\in \{0, 1, 2\}$：每条边细分点数量。
- Layer Subdiv Offset $p \in [0, 0.5]$：细分点插值比例（$0.5$ 为中点；$p$ 与 $1-p$ 成对对称）。
- Layer Draw Rim：是否绘制本层环边（按顺序封圈）。
- Layer Draw Diagonals / skipK：是否绘制星形/对角线（跳步 $k$ 连接）。
- Inter-Layer 1-1：是否连接到内层最近的 $1$ 个点。
- Inter-Layer 1-2：在 $1\to1$ 基础上再连接其相邻点之一（总计两条）。
- Inter-Layer Zigzag：若开启，两条邻边按外层点奇偶交替选择左/右邻点。
- Jitter Deg / seed / Connect to Center：角度微抖动、随机种子、以及中心点是否必连最内层。

**生成方式**对第 $l$ 层：

1. 生成正 $n_l$ 边形顶点（半径 $R = R_l$，带相位偏移）。
2. 沿每条边 $\overline{AB}$ 追加细分点（线性插值）：
$$\text{subdiv} = 1: \ (1-p)A + pB \quad (\text{或}(A+B)/2),$$
$$\text{subdiv} = 2: \ (1-p)A + pB \text{ 与 } pA + (1-p)B,$$
二者关于边中点对称。
3. 按边顺序将“端点 + 细分点”压入 `ringIds[l]`，并保存每个点的极角
$$\alpha = \operatorname{atan2}(y, x)$$
到 `ringAng[l]` 作为层间匹配依据。

**连边规则**

- **层内环边**（`drawRim=true`）：按 `ringIds[l]` 顺序相邻相连并封圈。
- **层内对角/星形**（`drawDiagonals=true`）：以跳步 `skipK` 在同环内连接。
- **层间 $1\to1$ 最近连接**：外层点连接到内层角度最近点（用 $\alpha$ 近邻匹配）。
- **层间 $1\to2$ 扩展**：再连接该最近点的左/右邻居之一。
- Zigzag **交替**：若开启，外层点按奇偶交替选择左/右邻居，形成交错网格效果。
- **中心连接**：中心点与最内层所有点相连（若 `Connect to Center` 启用）。

**复杂度分析**

- 点生成：$\mathcal{O}(\sum_l n_l \cdot (1 + \text{subdiv}_l))$
- 层内环边/对角：$\mathcal{O}(\sum_l M_l)$，其中 $M_l$ 为第 $l$ 层实际点数。
- 层间连接（相邻两层）：$\mathcal{O}(M_l \cdot M_{l+1})$

## Insights

在本项目中，我体会到：参数化生成可以让复杂结构被压缩为简洁的少量可控变量，而 Qt 的 Graphics View 框架则天然提供了稳定的渲染与交互基座。通过将节点与边的生成逻辑与渲染逻辑解耦，能够快速迭代新的连接规则与视觉风格，并保持兼容性与可扩展性。

未来我希望继续引入更多生成范式（如力导向布局、约束优化），让本项目在可视化与设计探索中更进一步。

## Acknowledgement

特别感谢 Gemini 与 ChatGPT 系列大语言模型。作为我的变成助手，它们在 Qt 调试、CMake 跨平台构建配置以及本海报的文案润色中提供了关键协助，极大地加速了从灵感到原型的迭代效率。

## Who am I?

我是余政希，同济大学国豪书院 2024 级人工智能（精英班）本科生，热爱计算机与人工智能方向。入学以来，我积极参与多项科研课题与工程项目实践，注重将理论学习与真实场景应用相结合，持续提升专业能力与创新素养。期待在技术创新与实践探索中不断突破自我，为团队协作与共同成长贡献力量。

**Email:** 2452633@tongji.edu.cn

**Github:** Github/yzxoi