

3D Vibe Coding 手册

从一句话到能跑的场景：GPT-6 Astra × Tripo全流程实测

一个不会写代码的人，怎么用Tripo加一个编程Agent，把一张设计图做成能开车逛的巴黎、能拆能装的发动机、能走进去的星球——每一步都有真实截图，配置怎么选都写清楚。

3D Vibe Coding — From a Sentence to a Scene That Runs

信息来源：Tripo Studio与开发者文档一手截图 · 2026年9月本机实测（六个可玩demo、一套管线实验）· 官方定价页对账

文档版本：v260914c

发布时间：2026-09-14 (build #4)

涵盖内容：Tripo Studio全功能实操 · 提示词与设计图 · Three.js / Blender / Unity协同 · 巴黎来信全流程 · 三个教学演示案例 · 随书资源包 · 附录B踩坑索引（卡住了按症状查）

Tripo × 花叔

这条流水线上的活儿是这么分的：3D这一层——生成、减面、拆件、贴图、绑骨——全在Tripo Studio里跑；GPT-6 Astra只干两件事，它驱动的编程Agent替我写完了所有demo的代码，以及生成书里的设计图。它操作Blender这类专业软件的能力我没测过，别把我没做过的事算到它头上，完整口径见附录D。

花叔

AI Native Coder

这本书是我做视频时的实测记录，判断和翻车都没删。所有截图与数字来自作者本人2026年9月的真实操作；产品界面与价格随时会变，以官方页面为准。本文档在Claude Code辅助下整理编写。

目录

五部分二十三节，附录四篇；每节都能从「怎么做」直接跳到「为什么」。已经动手、卡在某个症状上，别从头读——直接翻附录B踩坑索引：Three.js、Unity、Tripo Studio、Blender、发布打包，五张表按症状查。PDF里这份目录、阅读器的侧边书签栏、以及正文里每一处「§ 12」「附录B」都是可点的，跟着引用直接跳，不用回目录查页码。

开始之前

§00 这本书怎么用+资源包	How to Use This Book, and What's in the Kit	4
----------------	---	---

第一部分 · 一张图到一个模型

§01 十分钟跑通第一次图生3D	Your First Image-to-3D in Ten Minutes	16
§02 参考图怎么准备	Preparing the Reference Image	33
§03 生成之后	After the Generate Button	37
§04 减面与重拓扑	Decimation and Retopology	49
§05 智能拆分与贴图生成	Part Segmentation and Texture Generation	69
§06 绑骨与动画	Auto-Rigging and Animation	84
§07 怎么把积分花在刀刃上	Spending Credits Where They Count	100

第二部分 · 提示词与设计图

§08 为什么先出设计图再建模	Design First, Model Second	110
§09 提示词技巧示例（含图）	Prompt Recipes for Image-to-3D	122
§10 批量出图与命名	Batch Generation, Naming, Retries	145

第三部分 · 接进工具

§11 把Tripo接进你的工具：五条路的入口与门槛	Five Integration Paths — Entry Points and Blockers	154
§12 Three.js：让模型在网页里跑起来	Three.js — From GLB Files to a Playable Page	172
§13 Blender：什么时候该进，什么时候别进	Blender — When It Earns Its 300 MB	196

§14	Unity：把资产搭成一座会动的园区	Unity — Ten Traps Between a GLB and a Playable Build	213
§15	谁负责哪一层	Who Owns Which Layer — A Relay, Not a Race	231

第四部分 · 做一款游戏

§16	从选题到资产清单	From Topic to Asset List	242
§17	生成与清洗流水线	The Generate-and-Clean Pipeline	255
§18	组装：把23件模型变成一个能开的城	Assembly: From 23 Files to a Drivable City	265
§19	验收与发布	QA and Release	283

第五部分 · 做教学演示

§20	机构类：一台能拆能装的发动机	Teaching Demos I: A Four-Stroke Engine You Can Take Apart	297
§21	艺术类：把一幅画拆成五层	Teaching Demos II: Unfolding a Painting into Five Layers	309
§22	世界类：一颗你走得完的星球	Teaching Demos III: Building a Small World You Can Live In	321

附录

附录A	资源包索引	Appendix A · Resource Kit Index	334
附录B	踩坑索引	Appendix B · Pitfall Index	361
附录C	配置与积分参考	Appendix C · Configurations and Credit Reference	371
附录D	术语：用人话讲一遍	Appendix D · Glossary in Plain Language	381

\$00 这本书怎么用+资源包

How to Use This Book, and What's in the Kit

这一节回答四件事：这本书里有什么（三百八十多页、23节加4个附录）、动手之前先得知道什么（去哪注册、免费档够不够用、机器上得先装哪几样）、按你自己的目标该从哪一节翻起，以及随书资源包里有什么、缺什么。五分钟读完这节，你就知道剩下那几百页跟你有没有关系。

先看东西

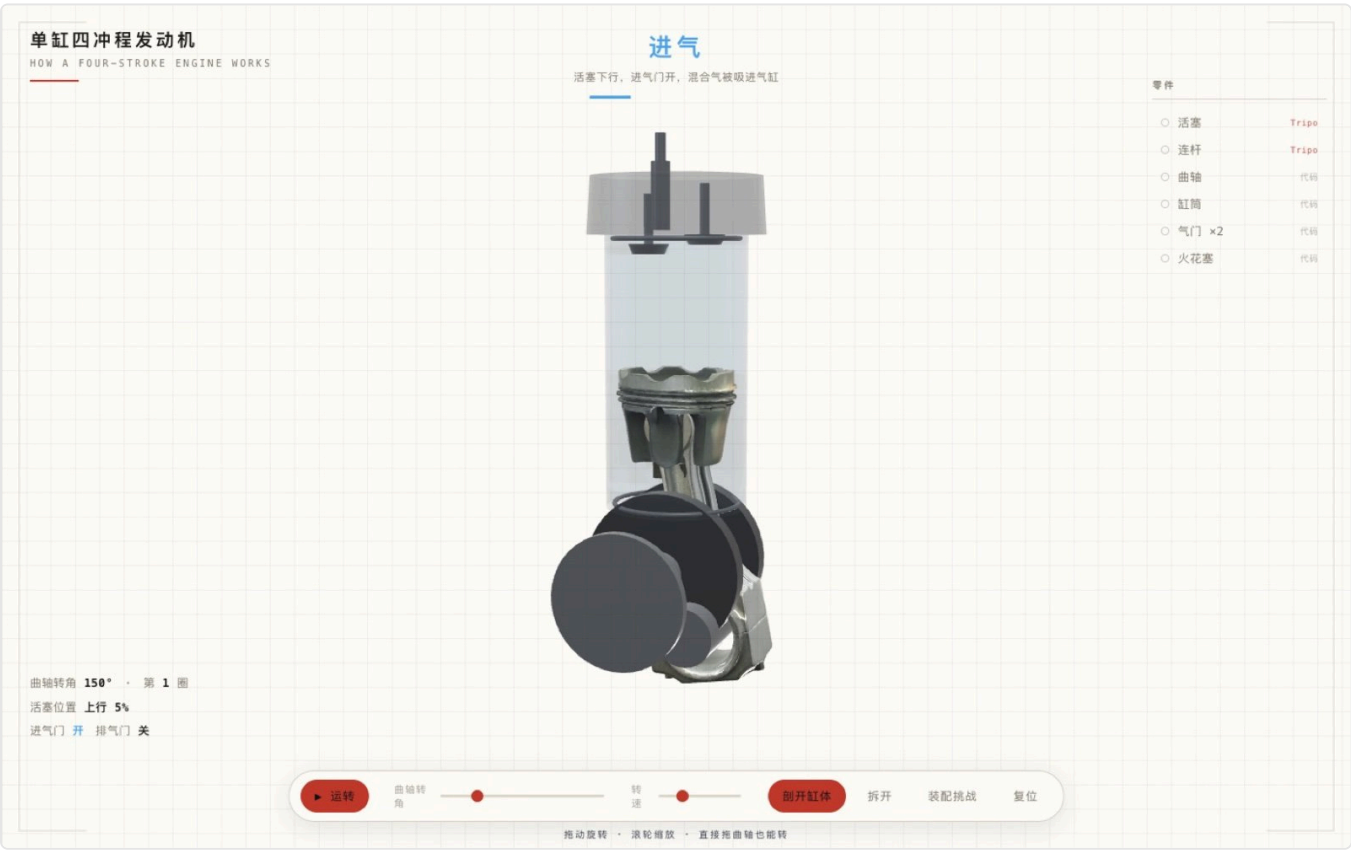
三张图。一张是能开着老爷车在塞纳河岸上跑的巴黎街区，一张是能走到星球背面去的小王子，一张是能剖开缸体看活塞上下的四冲程发动机。



「巴黎来信」demo的驾驶视角，2026-09-11复核时截的。画面里的奥斯曼建筑、行人、行道树和这辆车全是Tripo生成的资产，跑在Three.js里，左下角是程序化生成的街区地图。



小王子星球demo的A视角（原作星球），2026-09-11复核截图。人物用的是Tripo自动绑骨给的41关节骨架，星球表面是一张等距圆柱贴图而不是建模出来的地形。



四冲程发动机demo的进气冲程，2026-09-11复核截图。注意右边那张零件表：标「Tripo」的活塞和连杆是生成的，标「代码」的缸筒、气门、火花塞是agent用几何体搭的。这本书有一半内容在讲这条分界线画在哪。

这三样东西是2026年9月那两周里真做出来的，能点开玩、能拖动、能拆。我不会写代码（所有产品都是AI写的，这是老读者都知道的事），3D更是第一次碰。这本书就是把那两周走过的路原样写下来，每个按钮、每条命令、每次翻车，都带着截图和积分账。

给谁看

你在用Codex、Claude Code、Cursor里的任意一个，做过点东西出来，但没碰过3D，也不打算去学建模。你想做的是能给别人玩的东西：一个网页小游戏、一段科普演示、一个能转能拆的产品说明。

不是给3D美术看的。书里所有「怎么建模」的部分都交给了Tripo，所有「怎么写代码」的部分都交给了agent，我干的事情只有两件：**决定哪一层归谁**，以及在它们做砸的时候知道是哪一步砸了。这两件事正好是AI替不了的，也是这本书真正的内容。

这本书里到底有什么

先给你一个体量的概念，免得你以为这是一份十几页的速查卡。

项	数	说明
篇幅	386页	A4版式，含图，最后一页是附录D的术语表
正文节数	23节	§ 00到 § 22，分五个部分
附录	4个	A资源包逐件索引 · B踩坑索引 · C配置与积分参考 · D术语表
配图	几乎全是真实截图	Tripo Studio的每个面板、每次生成的读数、每一笔积分，都是我自己点出来截下来的

三百八十多页这个体量有点吓人，所以先说清楚它的性质：**这是一本查的书，不是一本读的书**。五个部分之间是并列关系不是递进关系，你从哪一部分切进去都能用。真正需要顺着读的只有Part IV那四节，因为那是一个项目从选题到发布的完整时间线。

我自己检验一本工具书有没有用的标准只有一条：**你照着做，能不能得到和书里一样的数**。所以书里每个数字都带出处，每个界面都带截图日期，翻车的部分一个没删。

五个部分，各解决什么

部分	替你回答的问题	节
I · 一张图到一个模型	生成表单上那一排开关该开哪个、一次生成为什么扣50积分、表单里默认的Polycount 20000怎么决定了出来的模型有多少面、太重了怎么压	§ 01– § 07

部分	替你回答的问题	节
II • 提示词与设计图	为什么先让agent画一张设计图再去建模、那句提示词长什么样、怎么一次批出十几件风格统一的资产	\$ 08– \$ 10
III • 接进工具	模型拿到手之后，API、CLI、Codex插件、Blender、Three.js、Unity各自在哪一层接手，谁不可替代	\$ 11– \$ 15
IV • 做一款游戏	23件资产怎么定清单、怎么批量清洗、怎么指挥agent把它们装成一个能玩的东西并发布出去	\$ 16– \$ 19
V • 做教学演示	机构演示、名画演示、故事世界三类各自卡在哪：运动学、风格保真、贴图显存	\$ 20– \$ 22
附录	资源包逐件索引、踩坑索引、价目表、术语表	A–D

开始之前：注册、免费档、什么时候才该订阅

书里的截图全是我自己那个付费账号，余额两万多，你别照着它估成本。真要动手，先把下面五行看完，五分钟就够。

问题	答案
去哪注册	<code>studio.tripo3d.com</code> ，注册完直接进工作台，\$ 01那张首页图就是它
免费档给多少	200积分。官网自己折算的是「约13个模型」——那是按几何底价15算的；按界面默认那一档50算，是4个
200积分够走完\$ 01吗	够，而且不止一次。\$ 01从头到尾只用「生成一个模型」这一个动作，工作台默认档一次50积分（首页那个快捷入口是55，差价在\$ 01里讲了）
什么时候才需要订阅	三件事从Pro起才有：模型私有、可商用、DCC Bridge（浏览器一键推进Blender / Unity / Unreal）。Free那一档 Private Models 和 Commercial Use 官方对照表里都是×，Exports 写的是 15 (H2.5 only)
价目在哪看	<code>tripo3d.com/pricing</code> ，拉到底FAQ第一条展开就是官方积分表。逐行读数和截图在\$ 07与附录C

截至2026-09官方标价与界面实测，以官网为准；最新价看 `tripo3d.com/pricing`。

一句话的判断法：免费档帮你决定要不要付钱，帮不了你做出一件能交付的东西。先用它把生成质量看准，再决定订不订。

开始之前：机器上得先有什么

上面那张表管账号，这张表管机器。这些东西书里是分散在各节里交代的（Node在 § 11、Blender在 § 13、Unity在 § 14），第一次读容易走到半路才发现缺东西，所以在这里集中列一次。只想在浏览器里点几下生成模型的话，一样都不用装——下面这些是「要把整本书跟着做完」的清单。

装什么	版本要求	哪一节会用到	不装会卡在哪
Node.js	官方要求 20或更新 ；我这台跑的是v26.0.0（§ 11记了完整环境）	§ 11的CLI与Codex插件；§ 04 § 12 § 13 § 14的减面降贴图命令	所有 <code>npm / npx</code> 开头的命令都跑不起来。终端敲 <code>node -v</code> 先验一次
@gltf-transform/cli	书里实测4.5.0。不单独装也行，命令前面补 <code>npx --yes</code> 每次现拉，只是多等几秒	§ 04减面、§ 12降贴图、§ 17批量流水线	减面和降贴图这两道没有替代路径。它依赖Node，上一行先满足
Blender	DCC Bridge官方要求 4.1.0或更新 （§ 11引的官方博客）；书里全部实测跑在4.4.3	§ 03那条无头命令、§ 13整节，以及脚本 <code>fbx2gltf.py</code> 和 <code>render_gltf6.py</code>	FBX转GLB和六机位渲图两条路断掉，§ 13只能读不能做。注意 § 13另有一坑：4.4的glTF导入器不认 <code>EXT_meshopt_compression</code> ，带这个扩展的GLB一个顶点都进不来
Python 3	资源包 脚本/ 里10个 <code>.py</code> 。 <code>gltf_downscale.py</code> 要 <code>numpy</code> ； <code>fbx2gltf.py</code> 要 <code>numpy</code> 和 <code>Pillow</code> ； <code>selfcheck_props.py</code> 要 <code>playwright</code> （走本机Chrome，不另下浏览器）	§ 03 § 04 § 10 § 12 § 13 § 17 § 19；起本地服务的 <code>python3 -m http.server</code> 也是它	那批脚本跑不了。而且没有本地HTTP服务，双击用 <code>file://</code> 打开的3D页会被CORS挡住、GLB根本加载不进来（附录B第一条）
浏览器	DCC Bridge官方只支持 Chrome 116+、Edge 116+、Opera 102+ （§ 11）；跑demo和量GPU耗时还要WebGL2	§ 11 § 12 § 19	DCC Bridge那个一键推送按不动；§ 12量每帧GPU耗时的那个扩展挂不上，只能退回看帧间隔（被vsync锁住，量不出差别）
Unity （只做 § 14 才要）	Unity 6000.0.83f1 + glTFast 6.0.1，内置渲染管线	§ 14	只影响 § 14，其余各节不装它也能整节做完（§ 15会谈它该接哪一层，但不用你打开它）。 Unity 6改了菜单名 ，照旧教程会找不到窗口，§ 14开头有对照

装什么	版本要求	哪一节会用到	不装会卡在哪
ffmpeg（只做 § 14 末尾那段片子才要）	无特殊版本要求	§ 14 把逐帧图拼成巡游片那一步	模型和场景都不受影响，只是最后合不出视频

还有一件必须先说的：全书的命令和路径都是在 macOS 上跑出来的。 机器是 Apple Silicon 的 Mac，系统 macOS 26.6.0。所以你会看到 `/Applications/Blender.app/Contents/MacOS/Blender --background` 这种写法——那是 Blender 可执行文件在 macOS 上的位置，Windows 和 Linux 要换成你自己那台的路径，**但命令的参数本身（`--background`、`--python`、`--factory-startup`）三个平台通用，换掉前面那一截就能用。** `npm`、`npx`、`glTF-transform`、`python3` 这些本来就不带平台路径，照抄即可。**我没有 Windows 机器，也没在 Linux 上验过，所以这本书里不会出现一条我编出来的 Windows 路径**——编一条让你照着敲，比不给更糟。唯一的例外是 § 11 那三条 CLI 的 Windows 注意事项，那是照抄官方文档的，我在原地标了「没验过」。

按需跳读

这本书是查的，不是从头读到尾的。按你现在卡在哪儿挑：

- 今晚就想生出第一个能用的模型：§ 01 把表单讲完，§ 03 讲怎么把文件拿到本地，两节够了
- 看完视频来找那四条参考图配方的：§ 08 开头就是，一个框，四条，加一段可以整段复制的提示词原文。想知道为什么是这四条、每一条各挡掉了什么，往下翻到 § 09
- 想直接抄提示词：§ 09。它是从我真跑过的 95 条提示词里挑出来的，大部分配「提示词原文+设计图+ Tripo 结果」三联，少数只有提示词原文，能照着改自己那条
- 被积分吓到了：§ 07 和附录 C。默认设置下一次生成 50 积分，其中 35 分买的是 8K 贴图；把贴图档位拨到 2K 是 30，只要几何是 15
- 模型太重，网页打不开：§ 04 讲减面和重拓扑，§ 12 讲贴图吃显存那件事（17 件资产 5.38 GB 显存，显卡工具按 1024 进位读作 5.0 GiB；降到 1024² 之后 0.22 GiB，折成十进制是 0.24 GB）
- 要让人物动起来：§ 06 讲自动绑骨的全流程，做不好再翻 § 13 看 Blender 里手工绑的对照组为什么更差
- 目标是一款能玩的游戏：§ 16 到 § 19 顺着读，或者倒着从 § 19 的验收清单读起，先知道终点长什么样
- 目标是一段教学演示：§ 20 到 § 22，三种类型挑最像你那个的
- 已经在做，卡住了：附录 B。一共 66 条，分五块——Three.js 与加载 GLB 23 条、Unity 13 条、Tripo Studio 与平台侧 21 条、Blender 与格式转换 6 条、发布与打包 3 条，渲染成黑、角色发白、一排人同步走这些都在里面
- 不知道该谁干哪一层：§ 15。如果只读一节，读这节

这本书和资源包怎么拿

这本书是免费的，跟着我那条 Tripo 视频一起送。领的方式只有一条路：**给那条视频一键三连，然后在评论区回复「3d」，我发给你。**

没做成一个随便点的置顶链接，是因为我想知道它到底被多少人真的拿走了，这个数会决定我下一本橙皮书写不写、写多厚。你回复的那两个字符对我是有用的信息。

发给你的是这本书的PDF，加上下面要讲的那套资源包。PDF适合在电脑上翻和搜，386页里每一页都有页码、每一节都有书眉，正文里每一处「§ 12」「附录B」都是可点的，跟着引用直接跳，不用回目录查页码。

微信读书版（EPUB）随后补。那一版会多一封「给读者的话」，其余内容完全一样。没有和PDF一起发，是因为EPUB要把这本书里那些宽表重排成窄屏能读的样子，是另一道工序，我不想赶工出一个在手机上糊掉的版本。

我在视频里说的是「模型、提示词、脚本全在里面」，这句话大体是对的，但有一处例外和两处时间差（一批资产和几个脚本没赶上封版），都写在下面「包里没有什么」那一小节。**先读那一小节再解压，比解压完再回来找我省事。**

随书资源包有什么

先说盘点口径，免得你以为拿到的是个几百件的素材库。这轮项目里一共140个GLB文件，按模型内部的任务ID去重之后，**只有62次真正独立的Tripo生成**，其余都是同一件的减面版、降贴图版和引擎工程里的拷贝。从这62次里挑出可以再分发、且设计图和提示词都在本地的，最后进包**56件、81.1 MiB**，覆盖建筑、载具、人物、道具、植被、机械六类。

先把单位说死，免得你对不上数。全书只有一条规矩：**按1024进位量出来的读数写MiB / GiB / KiB**，自己按十进制算出来的写**MB / GB**（这条规矩在§ 12立的，那儿还解释了为什么显存那笔账按十进制算）。所以这一节里的文件体积一律写MiB（1 MiB等于1,048,576字节），和附录A那张逐件表用的是同一把尺子；macOS的访达按1000进位显示，同一批文件它会写成85.1 MB，**不是拷贝出了问题**，Windows资源管理器的读数则和这里一致。判据是「量的还是算的」，不是「文件体积还是贴图显存」——同一批文件也会以十进制出现：§ 12和§ 17那张38件的对照表写的是534.6 MB降到52.4 MB，对应的1024进位读数是509.9 MiB降到50.0 MiB（附录A那一组），同一批文件、同一个9.8%，只是两种进位。两种进位差多少是固定的（MB和MiB之间约4.9%，GB和GiB之间约7.4%），**所以同一笔账的倍数在哪一边都成立**，别拿两边的数直接相减。资源包的体积一律看MiB，那是文件本身的数。

挑的时候按两组对照来凑，不是按好看来凑：

两组对照：A组19件巴黎资产是1024²贴图的原件（26.4 MiB），B组是拿A组这19件在本地跑一道 gltf-transform optimize 之后的512²版（9.8 MiB）。**这一对不是「只降贴图」的纯对照：**那条命令默认连一遍轻度简化一起跑，所以三角面也从345,935降到了338,924（-2.0%）。16.6 MiB的降幅里，13.3 MiB是贴图、3.3 MiB是那2%的面加上顶点量化，**大头在贴图**。要一个几何完全没动的纯对照，看附录A的G组，那38件降贴图前后三角面逐件相等。E组三件带41关节骨架和一段2.38秒的行走动画，F组三件是同一个模型走Blender、命令行、Blender绑定三条路的产物，配着§ 13读才有意义。


```

tripo-vibe-coding-资源包/
├── 模型/
│   ├── A-巴黎街景-1K/          19件· 26.4 MiB · 10242贴图
│   ├── B-巴黎街景-512/        19件· 9.8 MiB · 与A一一对应
│   ├── C-巴黎植被与街具/      6件· 8.7 MiB · 梧桐树三态、喷泉、莫里斯柱
│   ├── D-游乐设施-8K原档/     6件· 30.8 MiB · 没优化过长什么样
│   ├── E-绑骨与动画/          3件· 3.9 MiB · 41关节+ preset:biped:walk
│   └── F-三模式对照/          3件· 1.5 MiB · 配§13读
├── 设计与提示词/
│   ├── tripo-source/          19张PNG + jobs.jsonl
│   ├── tripo-source-2/        12张PNG + jobs.jsonl
│   ├── 游乐设施-参考图/      16张PNG + jobs.jsonl
│   └── 小王子-输入/          22张PNG + 5个jobs*.jsonl (只有图和词)
├── 脚本/
│   ├── E1-减面极限.py         按0.5/0.2/0.1/0.05/0.02逐档砍, 找每类资产的下限
│   ├── E2-批量流水线.py       按资产类型分配减面率, 25件跑一遍131秒
│   ├── optimize_assets.sh     只降贴图不动几何的那一版
│   ├── pull_glb.py            绕开「另存为」对话框, 走签名直链取GLB
│   ├── measure_glb.py         逐顶点套蒙皮矩阵, 量绑骨模型的真实包围盒
│   ├── selfcheck_props.py     无头Chrome逐件拍自检图, 顺手量比例和离地间隙
│   └── glb_downscale.py       直接改gLTF的JSON与BIN把8K贴图降到10242, 不走Blender (§12主角)
├── fbx2glb.py                 FBX转GLB, 带三通道包围盒交叉核验; 也管定yaw和烘变换 (§13)
├── render_glb6.py             六机位离线渲图, 用来定朝向、看贴图有没有贴歪 (§06 §12)
├── e10_topology.py            数FBX里每个多边形是几边形—E10那个79.51%就是它数的 (§04)
├── _probe_fbx.py              不依赖Blender直接解FBX数面, 是上面那个脚本的独立第二通道
└── demo/                     巴黎/发动机/星月夜的静态可玩版 (小王子那个不随包发, 理由见下面警告②)

```

这个包是2026-09-11装配的, 装完逐件核过一遍: 56件的三角面、顶点、贴图张数与尺寸、骨骼关节数、动画片段名、文件体积, 全部和附录A那张表对得上, 一件都没差; 每个文件的头四个字节都是 `gLTF`, JSON块能解析, 拷进包之后的SHA256和源文件一致。整包221个文件、解压后235.3 MiB, zip 212.3 MiB (访达和浏览器按十进制显示, 同样两个数在那儿写作246.8 MB和222.6 MB)。附录A那张逐件清单可以直接当验收单用。

怎么确认你手上那份就是这本书对应的那一版。解压之后核三样: 清单.md 第一屏写着「装配日期: 2026-09-11」, 模型/ 下面是56件, 脚本/ 里是11个文件 (其中5个是2026-09-13补进去的, 是哪五个见下面「包里没有什么」的第三件事)。三样都对得上, 这一节讲的就是你手上那份。对不上, 以包内 清单.md 和 SHA256.txt 为准——那两个文件是跟着包走的, 这一节是跟着书走的。

包里没有什么

三件事我提前说, 免得你解压之后觉得缺斤少两。

小王子那一组不随包发模型。设计和提示词全在 (22张PNG加5个 jobs*.jsonl), 照着能把这条流水线完整跑一遍, 但GLB和那个可玩demo都撤了, 理由在下面警告②。「书里讲了、包里没有」一共两处, 这是第一

处，另一处是紧接着要说的那批P2.0资产；除掉这两处，书里提到的每一件你都能在 模型/ 里找到实物。

2026-09-12那批P2.0新资产，暂时也不在这个包里。那两天我又用智能网格模式跑了一批巴黎扩展资产（glb-1024/ 40件、glb-1024-新/ 13件，合计71.2 MiB），书里 § 06、§ 12、§ 17会拿它们当案例讲工序和数据，但资源包是09-11封的版，它们没赶上。**这一版里没有它们，这句话是确定的**（将来补不补是下一版的事，补了我同步改附录A那张「随包 / 不随包」的标记）。好在这不影响你照着做：§ 12和§ 17那条降贴图工序的输入只要是8192²原档就行，**包里D组那6件游乐设施正是没优化过的8192²原档**，拿它们跑 脚本/glb_downscale.py 能把同一条工序原样走一遍，数不会和书里的38件一样，但每一步的做法和核验项是同一套。



说的就是这一批：glb-1024/ 那40件的总览图，每件下面标着文件名和三角面数（11k到32k不等）。三座桥、十栋建筑与地标（歌剧院、先贤祠、卢浮宫玻璃金字塔、圣母院西立面、红磨坊、奥赛大钟…）、两艘塞纳河上的船、公交车警车送货车Vélib、七个各不相同的行人，加上街具和两棵七叶树。这一批走的是Tripo的智能网格模式，出来是四边面，和上面A到F六组的v3.1高精度件不是一套东西。

顺带把这一批的一个数放在这儿，因为它解释了为什么书里到处都在讲降贴图：P2.0出来的贴图是8192×8192的，解码进显存之后单件要358 MB，26件摆进同一个场景就是9.3 GB，显存会炸。降到1024²之后单件5.59 MB、26件145 MB，几何一个三角面都没动：那一轮核过的38件，合计735,232个三角面，降贴图前后逐件相等，省下来的全是图片字节（534.6 MB降到52.4 MB）。全书这笔显存账统一按每像素4字节（RGBA32）展开再乘整条mipmap链的1.33倍算，1 MB当100万字节；你拿显卡工具去对，它按1024进位，读数会比这里小约7%（同一个数在那边是341 MiB、8.67 GiB），不是算错了。怎么做的在 § 12和 § 17。

第三件是反过来的：有五个脚本比书晚到，现在补上了。书里 § 04、§ 12、§ 13、§ 17反复在用 glb_downscale.py（降贴图、转朝向、烘变换）、fbx2glb.py（FBX转GLB，带三通道包围盒交叉核验）、render_glb6.py（六机位离线渲染图定朝向）、e10_topology.py（数FBX里每个多边形是几边形，解析器和_probe_fbx.py 是同一套），第一版装包时它们没赶上，等于教了半天工具拿不到手。现在五个都在 脚本/ 里，

进包前逐个体检过：没有绝对路径、没有密钥、没有账号信息。其中 `fbx2glb.py` 和 `render_glb6.py` 要本机装 Blender（它们是用 `--background` 模式调它）。下面那张参数表讲的就是 `glb_downscale.py`，配着 § 12 那五道工序和那份八项核验清单一起读。

参数	干什么
<code>glb</code>	输入GLB，位置参数
<code>-o / --out</code>	输出GLB，必填
<code>--texture-size</code>	贴图边长，默认1024；填0就是不降
<code>--jpeg-quality</code>	降采样后的JPEG质量，默认92
<code>--yaw</code>	绕glTF的+Y轴偏航多少度
<code>--scale</code>	等比缩放系数
<code>--match-height</code>	给一个旧GLB，自动按「旧件高度÷新件高度」算出缩放系数
<code>--keep-double-sided</code>	保留源文件的 <code>doubleSided</code> ；不加这个就强制单面
<code>--bake</code>	把yaw和scale烘进顶点数据，而不是写进根节点矩阵
<code>--emulate-node-scale</code>	配合 <code>--bake</code> ，把总变换拆成「根节点均匀缩放」加「烘进顶点的剩余部分」，世界包围盒不变
<code>--dump-texture</code>	另存一份降采样后的贴图，供肉眼检查
<code>--json</code>	把逐项核验结果写成JSON

脚本那一栏值得单说

资源包里最能直接省你时间的其实不是模型，是 `E2-批量流水线.py` 和 `optimize_assets.sh` 这两个：前者按资产类型给不同的减面率（人和车敢砍到0.25，建筑0.45，树和铁塔只敢0.75），25件跑完131秒、平均5.2秒一件；后者只resize贴图、一根三角面都不动，因为瓶颈从来就不在几何上。

GLB怎么打开

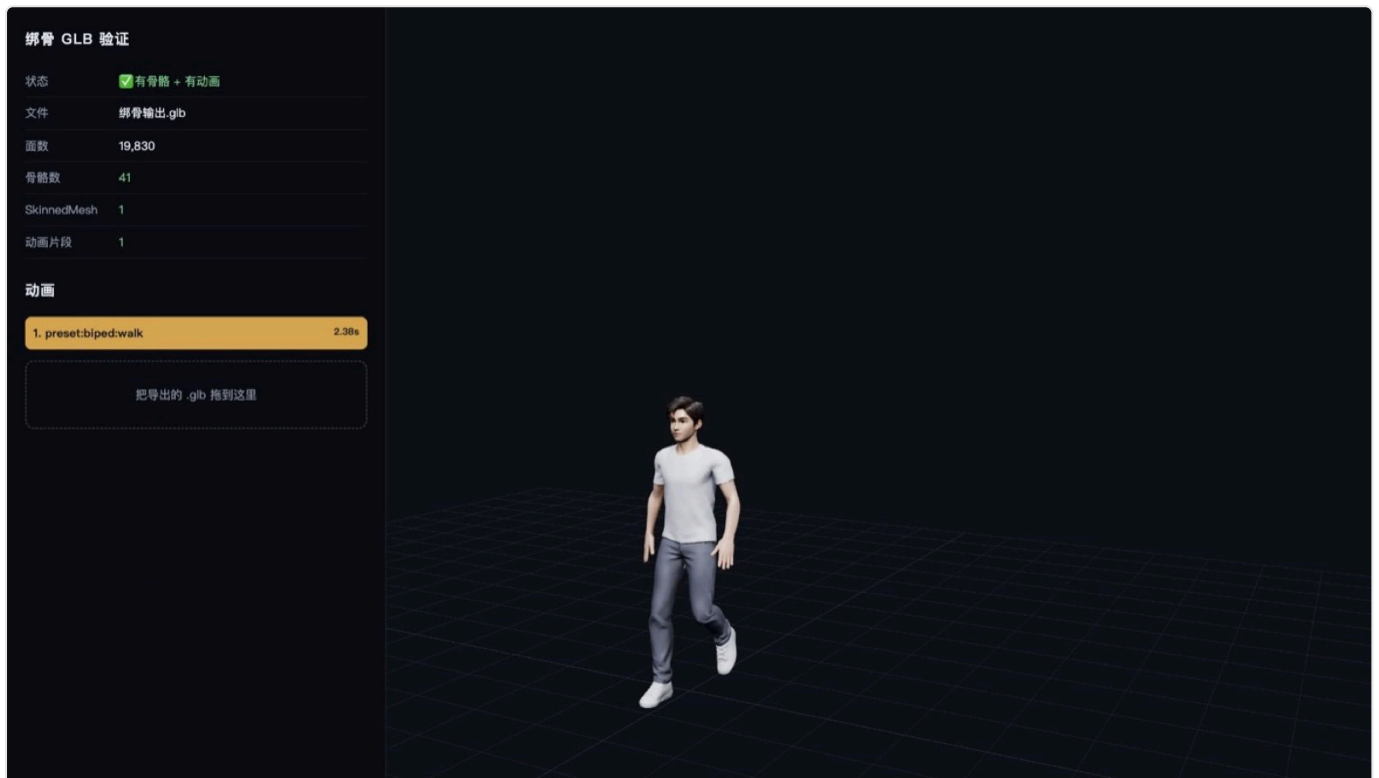
资源包里的模型全是 `.glb`。它是glTF这套3D格式的二进制打包版，网格、材质、贴图、骨骼、动画全塞在一个文件里，拷来拷去不会丢贴图。有四条路可以看它，按「花多久」排：

1 双击（macOS，零成本）

系统自带的「预览」把 .glb 和 .gltf 登记成了自己的默认文件类型，双击直接开，空格快速预览也走同一条路。看一眼形体对不对够用了。

2 拖进浏览器（最常用）

公开的在线查看器里我常开 <https://gltf-viewer.donmccurdy.com/> 和three.js官方编辑器 <https://threejs.org/editor/>，拖文件进去就行。不过做绑骨的时候我干脆让agent写了个二十行的验证页，因为在线查看器不会把我关心的那几个数直接摆在脸上。



自己写的绑骨验证页（2026-09-10绑骨实测时截）。把导出的.glb拖进虚线框，左栏直接报面数19,830、骨骼41、动画片段1条 preset:biped:walk 2.38秒，右边小人立刻走起来。这类一次性小工具让agent写比找现成的快。

3 传回Tripo Studio（想继续加工时）

Studio右侧资产栏顶部有个「上传3D模型」，接受OBJ / FBX / STL / GLB，单文件150 MB以内。传上去不只是看，拆件、绑骨、换贴图这些工具都能直接在它上面接着跑。

导出弹层长什么样、Format和Texture Resolution怎么选，§ 03有那张图和完整说明；左边黄色的「Send To」是直推Blender的出口，走§ 11。

4 Blender（要动手改的时候）

File → Import → glTF 2.0。只有需要改几何、烘焙贴图、修骨骼权重时才进这一层，什么时候值得进、什么时候进了纯属浪费时间，§ 13有实测。

许可：先看完这段再用

资源包里的模型是我用自己的Tripo付费订阅账号生成的，喂进去的设计图也都是我让agent画的，不含第三方照片、名画扫描件和品牌商标（这句管的是图和模型这两层，提示词那一层另说，见本段末尾和§09第03条）。挑件的时候剔掉了四类有风险的：在产商标车型（雪铁龙2CV）、名画高清复制品裁切（那份复制品的扫描版权没核）、维基照片裁切的发动机与木马、以及和已收录件同源只换了贴图档位的重复文件。**第三类要多说一句，因为容易被读成前后矛盾。**那两张图当初在维基图页上看到的标注就是公有领域，§02和§20写的是我当时用它的情形；但我没有逐张回去核对每张图的许可证具体是哪一版、有没有附加条件。「我自己拿它生成一个模型」和「把成品随书发给几千个人」是两件事，后者的标准更高——所以这一类在再分发这一层剔掉，不代表当初用它有问题。逐条理由写在附录A末尾。这四类别的是模型文件本身，管不到你写提示词时拿哪个真实专名当形体的尺子——所以别把上面那句读成「提示词里也没有品牌名」：包里 `tripo-source/jobs.jsonl` 的提示词原文里是有的（「与经典雪铁龙2CV同一美术风格」「雷诺4CV造型」这两处），它们只负责把形状说死，没有挂到任何一件资产上，怎么用、用完之后该注意什么，§09第03条有一整条讲。还有个口径你得知道：上面说的剔除只管 模型/ 那六组素材。**demo/ 里那三个可玩版是完整的程序，它们要跑起来就得带着自己那份资产**——巴黎里有那辆2CV和铁塔，发动机的活塞连杆来自维基照片，星月夜的村庄柏树来自名画裁切。所以它们是「打开玩一玩、看看这套流水线最后能做成什么样」的成品，不是给你拆出来当素材用的。要素材请用 模型/ 那六组，那里面是干净的。

上面提到的那批09-12的P2.0资产，将来真要补进包也是同一把尺子量：全部是我自己的设计图生的，没有一张输入图来自照片或扫描件；那批里的公交车、警车、游船、共享单车，提示词最后一句全部写死了「车身上不写任何文字、编号与徽章标识」，所以模型上没有品牌字样可以被烘进贴图。下面警告ⓐ对它们同样成立。

注意

两件用之前请先知道的事：

ⓐ包里这56件，请按「学习与二次创作参考」用。拆开看结构、改着玩、摆进自己的练习场景、拿它对着书里那些数复核一遍，都是我做这个包的本意；**请别把整包原样再转发出去**，它跟着书走，谁想要都能从同一条路领一份。**要拿它们做商用，别以我这段话为准，以Tripo官方条款和你自己账号那一档为准：**商用权限跟的是订阅档位，官网对照表里Pro及以上的 Commercial Use 是✓、模型也是Private，Free那一档这两项都是×。查询入口是 `tripo3d.com/pricing` 拉到底展开FAQ，截图和逐行读数在附录C。「这个包能不能再分发」和「你自己生成的模型能不能商用」是两件事，别混在一起看——后一件跟这个包没有任何关系，只看你的档位。

ⓑ小王子那一组不随包发模型。原著在多数司法辖区已进入公有领域，但角色形象在多国仍有活跃的商标与形象权登记，「过了版权期」这一句挡不住。所以那一组只收22张设计图和5个 `jobs*.jsonl` 提示词，当一套完整的图生3D流水线示例来读，教学价值本来也在流水线上不在题材上。

最后一句关于口径。书里每个数字都有出处：积分数来自Tripo官网定价页和界面上的实际扣费，面数和贴图尺寸来自直接读GLB文件头（不是听工具汇报的），耗时来自脚本自己打的时间戳。没实测过的能力我会写明「未实测」。你照着做如果结果对不上，大概率是版本变了，不是我编的。

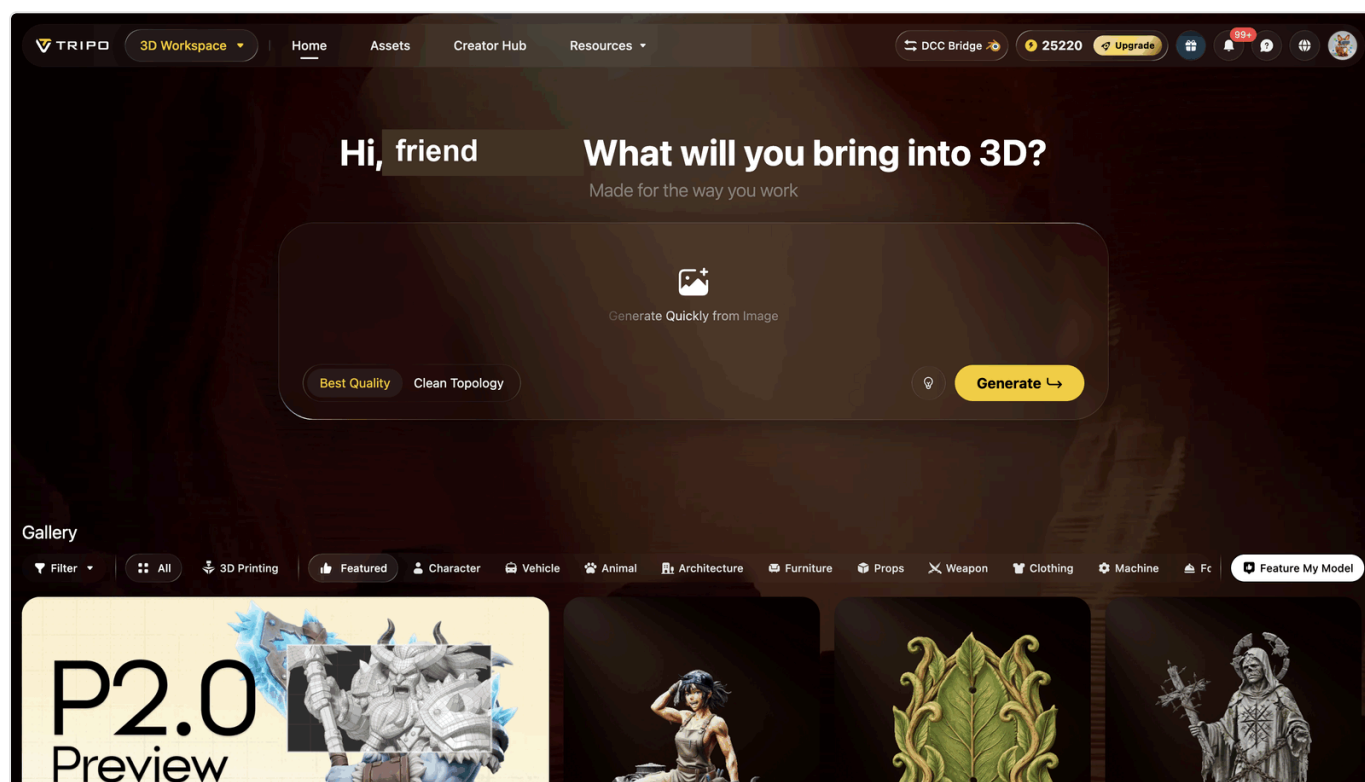
§01 十分钟跑通第一次图生3D

Your First Image-to-3D in Ten Minutes

这一节结束时，你会有一个能拖着转、能下载下来的3D模型，并且知道那个「生成」按钮上的数字是怎么算出来的、哪几个开关该关掉。

先看这一张图

Tripo Studio打开就是这个样子。整个平台你先只需要认识三样东西：顶栏右边那个闪电符号后面的数字（积分余额）、中间那个拖图进去的框、框下面那两个按钮。

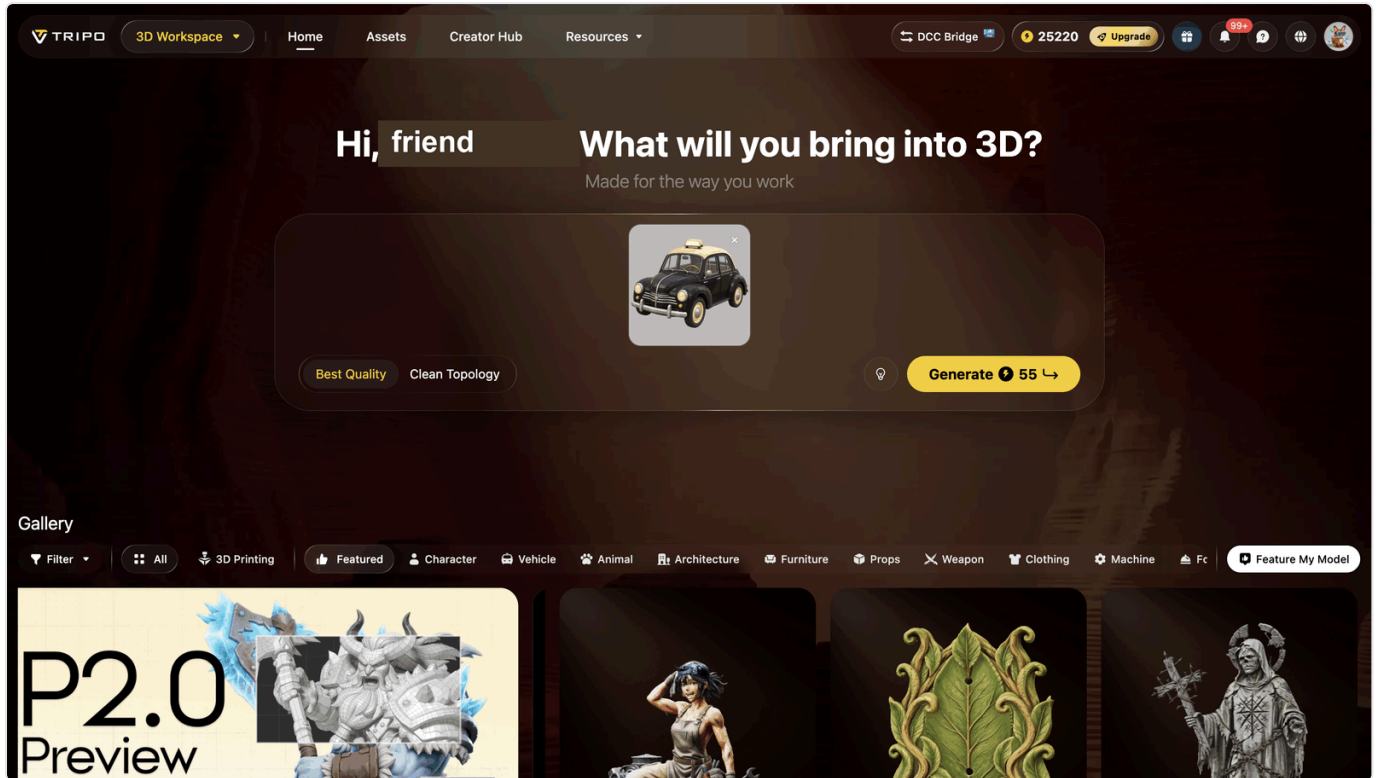


Tripo Studio首页 (studio.tripo3d.com)，2026-09-11。顶栏右侧「⚡ 25220」是账号的积分余额，中间的框可以直接把图拖进去，框下面「Best Quality / Clean Topology」是两种生成模式的快捷开关。下方Gallery是官方精选作品区。

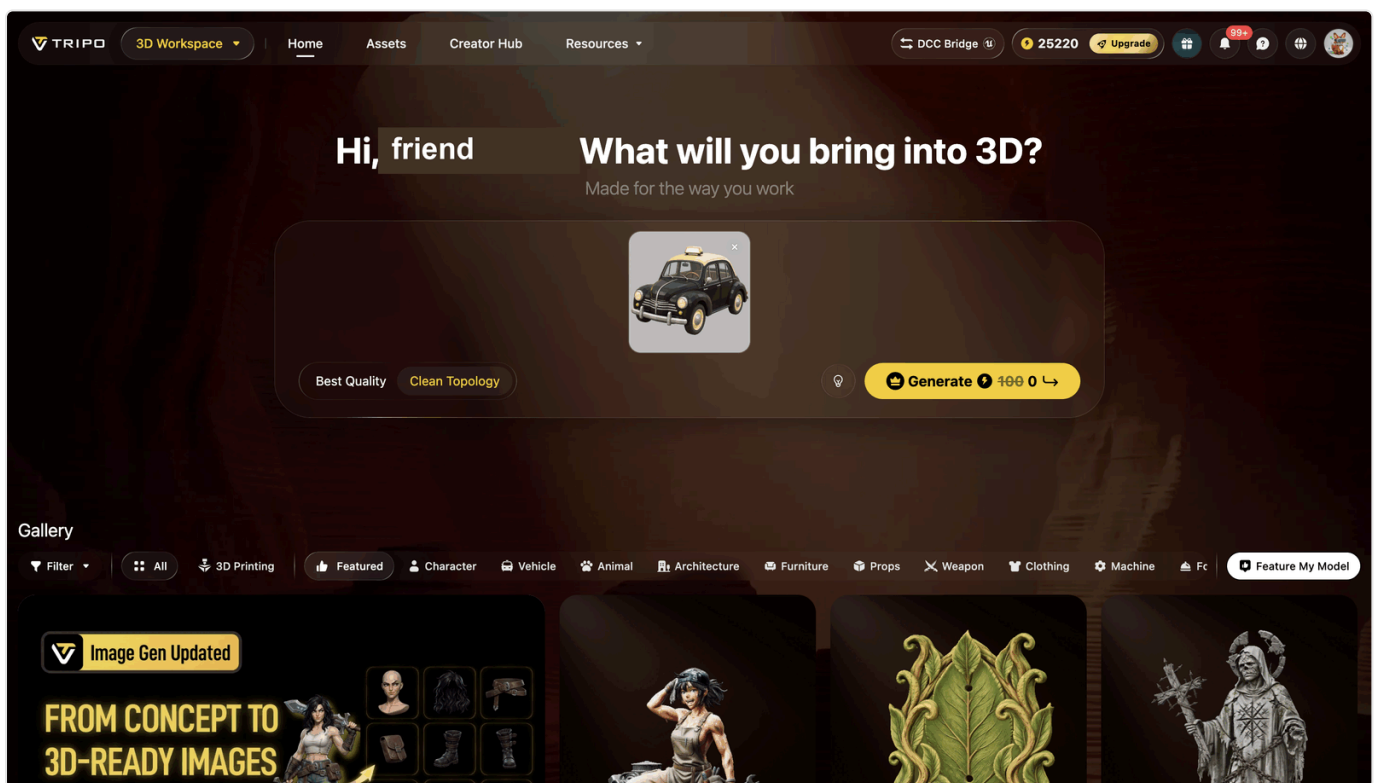
我的余额是两万多，你别照着这个数估成本。专业版订阅每月给3000积分，我这两万多是另外买的积分包。按官方标价，专业版¥140一个月3000积分，折下来一个积分不到五分钱；免费档给200积分，官方自己标的是「约13个模型」。

把图拖进去，价格立刻显示在按钮上

这是整个平台我最喜欢的一个设计：还没生成，按钮上就写着这一次要花多少。你不用去翻价目表，也不用生成完了才知道被扣了多少。



同一个首页，拖进一张1024×1024的出租车设计图之后。选中「Best Quality」，生成按钮变成「Generate ⚡ 55」。2026-09-11。



同一张图，切到「Clean Topology」。按钮上的100被划掉、后面跟一个0，这是P2.0预览期的一次性免费试用。2026-09-11。

首页这个框够快，但开关只有两个。真正干活的地方在工作台（顶栏左边「3D工作台」进去）。左边竖排一列工具：模型、智能拆分、重拓扑、贴图生成、动画，其中智能拆分和贴图生成下面还各有折叠起来的子工具。这一节只用「模型」那一个，剩下的后面各有一节。



工作台的「生成模型」面板，高精度模型模式，什么都还没传。左侧竖栏是工具列（模型 / 智能拆分 / 重拓扑 / 贴图生成 / 动画，智能拆分和贴图生成下面还有折叠项），面板里依次是输入区、通用设置（几何与贴图）、订阅专享（分部件生成 / 8K贴图 / 模型私密性「公开」）、AI模型（v3.1 – 最高质量，副标「预计等待更久」），底部「生成模型⚡ 50」。顶栏余额「⚡ 300」。国区站studio.tripo3d.com，2026-09-14。

两种模式：高精度模型和智能网格

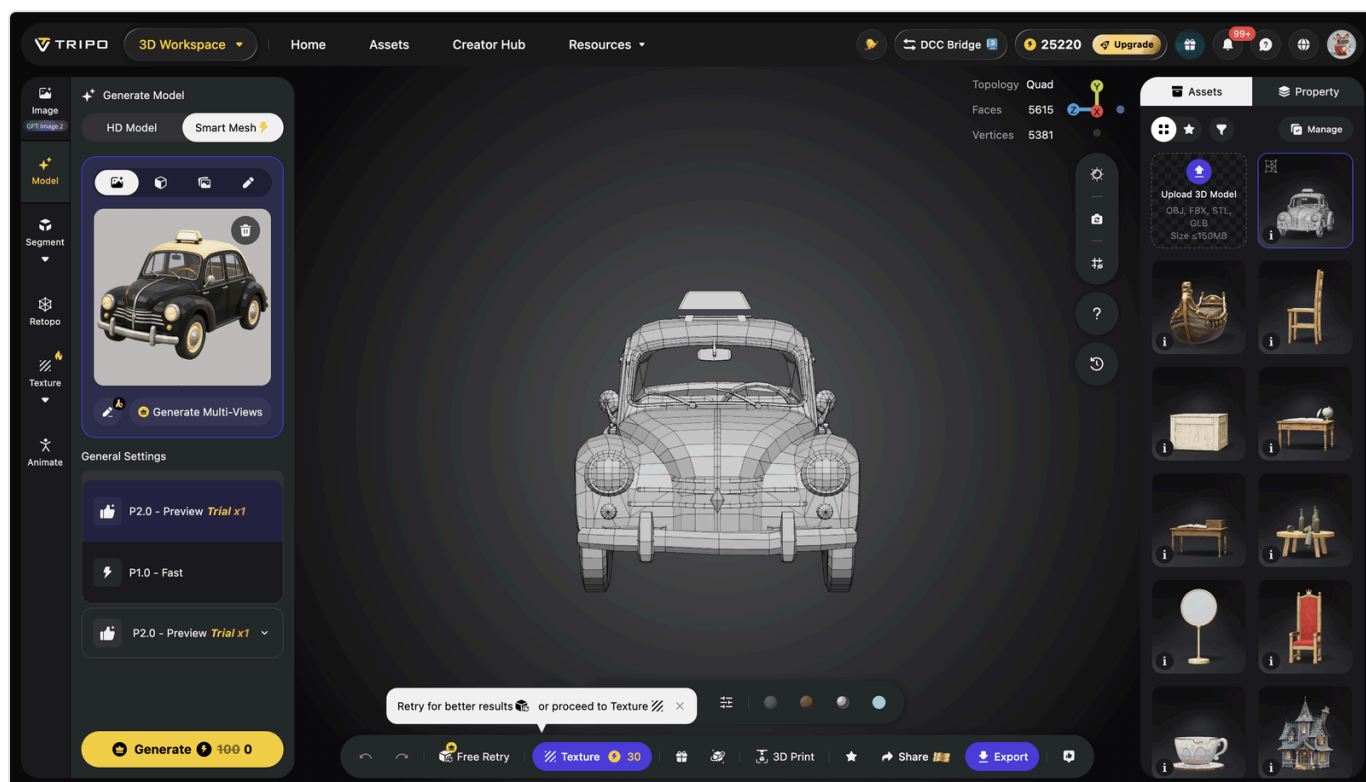
面板最上面那一排就是分岔口。这两个模式出来的是两种完全不同的网格，不分高下，各有各的去处。

	高精度模型	智能网格⚡
模型版本	v3.1 – Best Quality（界面自己标着「Expect longer wait times」）	P2.0 - Preview（试用一次） / P1.0 - Fast
默认拓扑	三角面	四边面
面数范围	500–2000000	四边面500–25000；三角面500–50000
贴图	直出就带	先出灰模，贴图另付30

价格	界面默认50（这是带8K贴图的价，只要几 何是15）	界面实扣65（原价100，划线打折）；贴图另付30，一件带色的 完整价是95。P2.0预览期首次免费
----	-------------------------------	---

表里的价格是截至2026-09我在界面上看到的标价和实扣数，以官网为准。

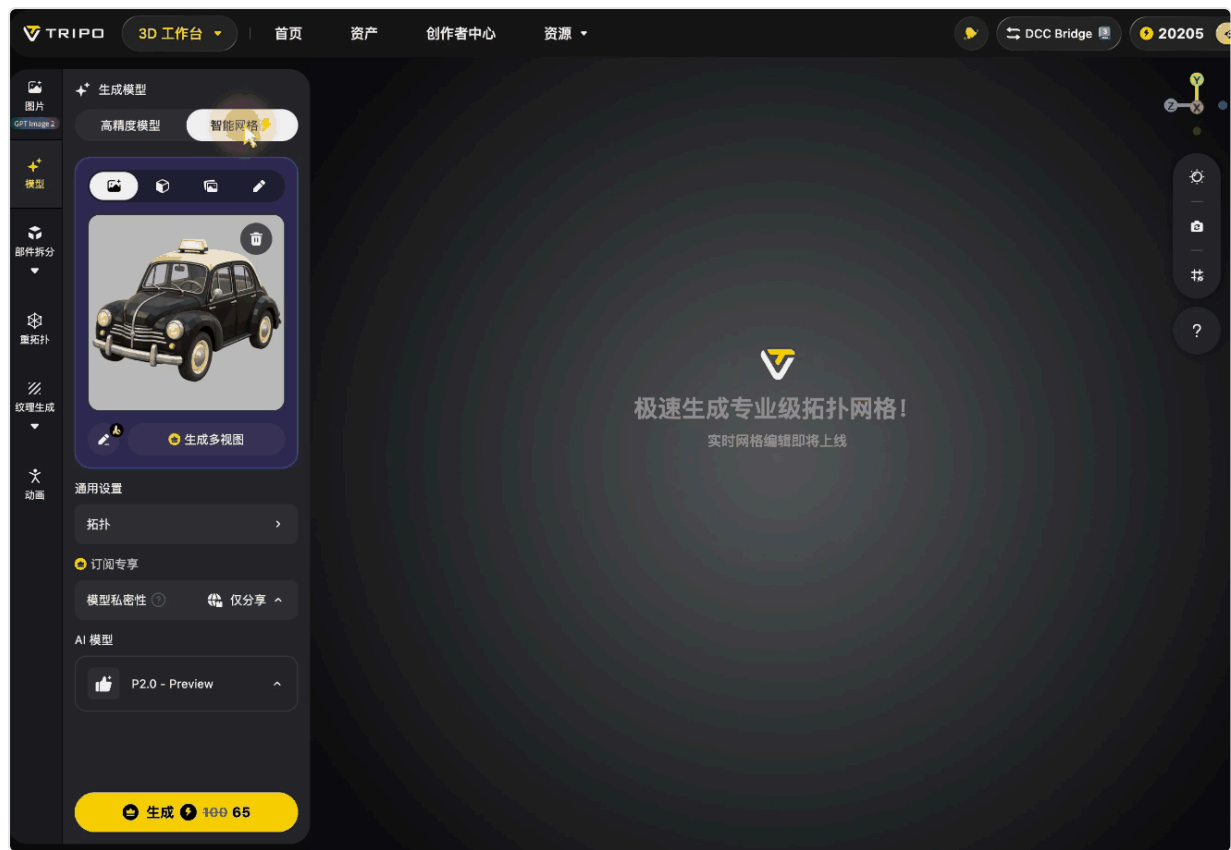
三角面和四边面：3D模型的表面是一小片一小片拼出来的。每一片三个角，叫三角面；四个角，叫四边面。引擎最终都按三角面渲染，所以三角面直接能用；四边面的好处是走线整齐，拿进Blender做修改、做形变的时候不容易撕裂，动画和雕刻这类活儿会舒服很多。



Smart Mesh模式下把AI Model下拉展开，只有两项：P2.0 - Preview（标着Trial x1）和P1.0 - Fast。右上角的读数是这次生成的结果：Topology Quad，Faces 5615，Vertices 5381。2026-09-11。

那次免费试用用掉之后，按钮上是什么数

上面那张图是2026-09-11，我的预览期试用还在，所以按钮写的是100划掉跟一个0。第二天再进这个表单，试用没了，我盯着按钮看那个数会不会回到100——没有，它是100划掉、后面跟一个65。原价划线、实扣65，之后我连着做的十四件，每一件都是按65走的。



中文界面的工作台，智能网格模式，AI模型选P2.0 - Preview，出租车参考图已经传进去。底部按钮「生成 ⚡ 100 65」——原价100被划掉，实扣65。顶栏余额20205。这一屏摘自2026-09-12的操作录屏，也是这本书里所有智能网格件的单价来源。

把这一件从头做完要花多少，拆开是这样的：**网格65，想要颜色再点一次贴图生成30，合计95**。如果只要几何不要颜色，高精度模式那边关掉纹理是15，智能网格这边是65——这两个数是纯几何的真实对比，不要拿默认的50去比，那50里有35分是8K贴图。

十四件乘65是910，这是我2026-09-12那天在网格上的全部开销。加上十三件贴图生成的390和一次自动绑骨的20，那一天一共1320积分。

面数那个框：两种模式数的不是同一种东西

这是我这两周最晚才搞明白、也最容易让人算错账的一件事。

两种模式的设置里都有一个定面数的框（高精度叫Polycount，中文界面叫「面数」），但这个数字在两边数的不是同一种面。高精度出的是三角面，那个数就是三角面的数；智能网格出的是四边面，那个数数的是四边面。而引擎最终只认三角面，一个四边面进引擎要拆成两个三角面。

拿同一张出租车设计图做过一次严格对照：两边都填8000，只换模式那一栏。

都填8000	高精度模型v3.1	智能网格P2.0
原生面	7,379个三角面	8,596个四边面
四边形占比	0%	79.51% (6,835个四边形)

都填8000	高精度模型v3.1	智能网格P2.0
三角化之后	7,379	15,431
顶点	3,646	8,580

同一个8000，智能网格拿到的几何预算是高精度那边的2.09倍。所以「同一张图、两种模式，外观差不多」这句话我原本想写，跑完这一组就划掉了——智能网格那版的保险杠栅条、大灯圈、门把手、雨刷全都在，高精度那版在同样的位置是几块糊过去的面。两边谈不上谁好谁坏，只是同一个数字在两边买到的东西根本不一样。

这组对照还有两个读数值得记下来。一是那个数是个目标不是硬上限：填8000，实得8,596个四边面，超了7%。二是四边面换算成三角面的倍数不是2，只有全是四边形的网格才趋近2。全书那批38件实测落在1.37到1.87倍之间，中位1.75，做面数预算就按1.75估；这一件8,596到15,431是1.795倍，落在分布偏上的位置。完整分布见附录D，实验过程、线框对照图和三条独立测量通道在 § 04。

核心建议

你在网上看到别人说「我用XX面数做出来的」，先问一句他的面数数的是三角面还是四边面。跨模式比面数之前先换算，不然结论会反。真正要控的预算是进引擎那一刻的三角面数，不是表单里填的那个数。

几何与贴图：那个弹层里每一个开关

高精度模型模式下点开「通用设置 → 几何与贴图」，会弹出一个七八项的面板。这是整个Studio里信息密度最高的一屏，也是最容易被别人一路默认点过去的一屏。



高精度模型的「几何与贴图」弹层全貌：超清几何精度（开）、AI补全&3D增强（开）、贴图生成（开）、贴图质量2K/4K/8K（选中4K）、PBR（关）、拓扑四边面/三角面（选中三角面）、面数控制滑块500–2000000（这一屏拉在上限2000000）。左侧订阅专享区：分部件生成（关）、8K贴图（关）、模型私密性「公开」。两件事先说在这儿：这一屏是国区一个免费档账号打开时的落位，不是我做这本书时的设法（我的设法见下表）；国区这个弹层上也没有单独对应「去除光照」的那一项。国区站studio.tripo3d.com，2026-09-14。

逐项说我是怎么设的：

开关	我的设法	为什么
Ultra Mesh Quality	关	加项，官方标价15积分。默认那些资产我这次全部没开，出来的质量做游戏和演示都够。
AI Complete	关	补全图里看不见的那一面。我给的设计图都是单物体白底、主体完整，没有需要靠猜的部分。
Texture	开	关掉就是灰模。除非你本来就要拿进Blender自己画贴图。
Texture Quality	我全程留着默认的8K，现在回头看该选2K	这一档是直接进价的：同一张图，8K比2K贵20积分（实测见下一节）。做网页和游戏，2K够用。
Remove Lighting	关	把参考图上的光影从贴图里剥掉。参考图本来就是柔和均匀布光的话，剥不剥差别不大。
PBR	开	加项，5积分。出来的模型带金属度和粗糙度，进引擎才有金属感。代价见下面那条警告。

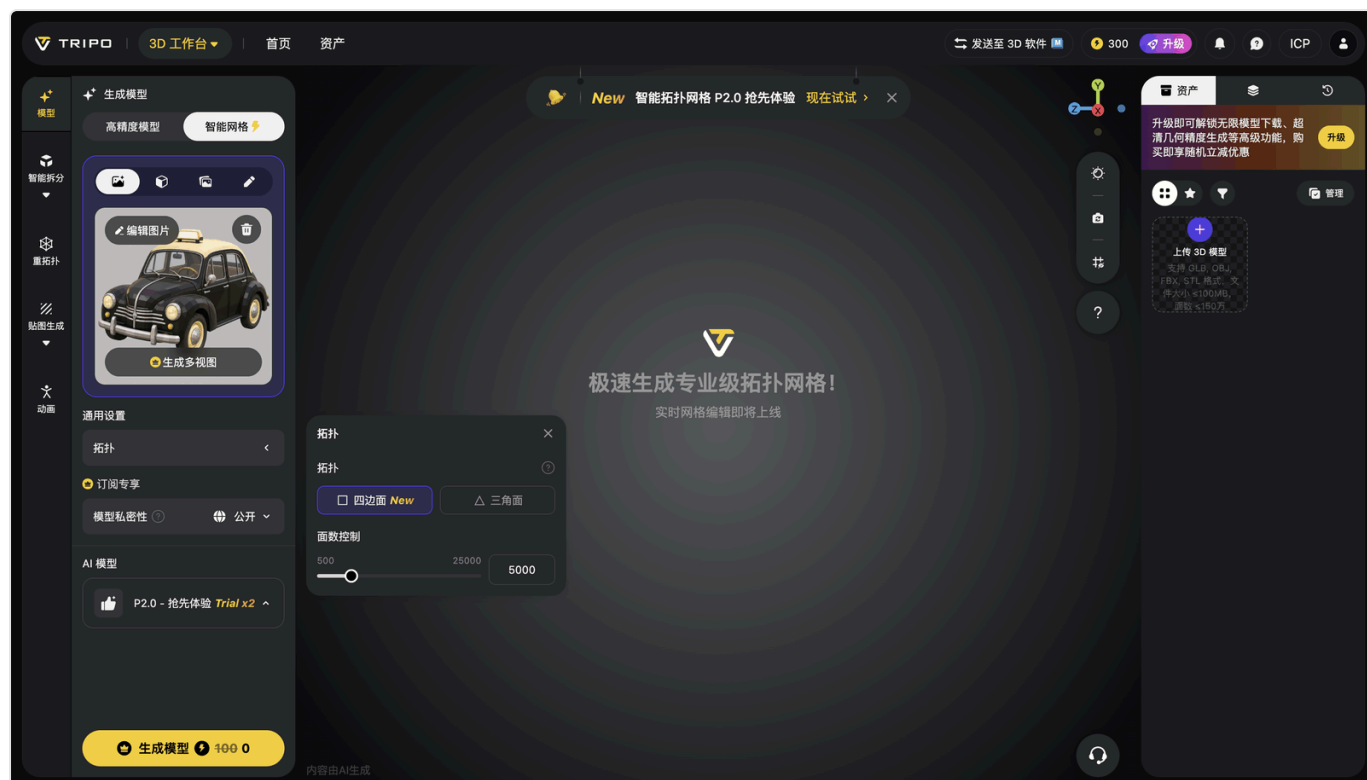
开关	我的设法	为什么
Topology	默认Triangle	Quad是加项，5积分。要注意Tripo的智能拆分不吃四边面模型。
Polycount	高精度这批我留了默认20000；后面走智能网格那批是一件一件设的	这是唯一能定面数的地方。你在参考图或者提示词里写「low poly」它不听，面数只认这个滑块。怎么分档见下面「一次批量」。
Generate in Parts	关（我有一次Trial没用）	加项标价30，生成时就按部件分开。
8K Texture	默认开	Members Only那一栏里的，跟上面的Texture Quality是同一件事的两个入口。
Privacy	Sharing Only	订阅用户才有的开关，控制生成的模型对外可见到什么程度。

注意

8K贴图是个甜蜜的陷阱，而且它收两道费。第一道是积分：8K那一档比2K贵20分，按下去之前就扣了。第二道是显存：Tripo默认导出8192×8192的贴图，一张底图摊进显存是358 MB，我做小王子那个demo时17个资产全按默认导，浏览器里量出来光贴图就占5.0 GiB（这是显卡工具那侧按1024进位的读数，换成本书算显存的十进制口径是5.38 GB，见 § 12），页面卡到没法看；统一降到1024×1024之后同样的场景只占0.22 GiB，游戏距离上肉眼看不出差别。

我这批资产两道费都交了：生成时留着默认8K，导出时又降到1k。做网页和游戏的话，生成就选2K，导出选1k，两头都省（导出弹层的Texture Resolution下拉见 § 03）。只有要拿去做特写、做印刷，高档才值那20分。

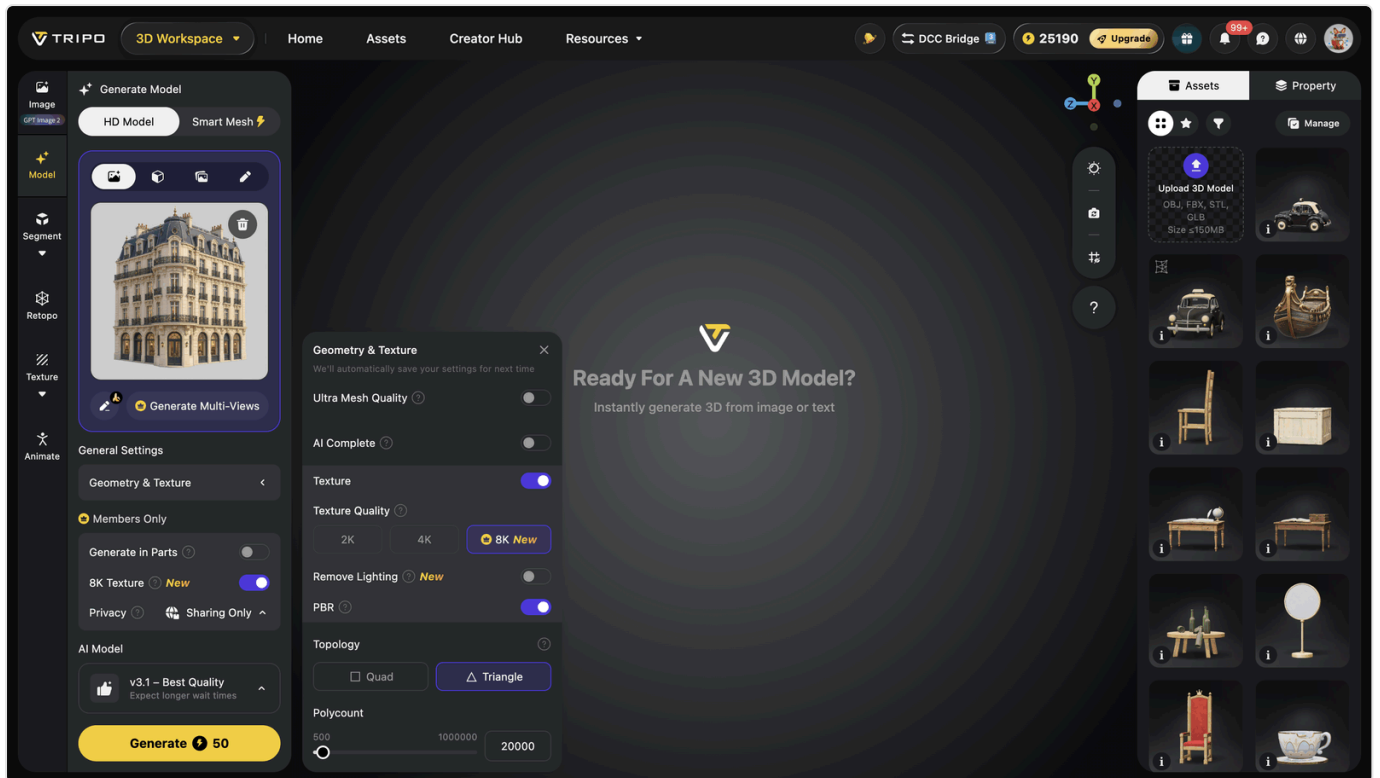
PBR: Physically Based Rendering，基于物理的渲染。简单说就是模型除了颜色，还带着「这块是金属、那块是磨砂」这类信息，光打上去才有正确的反应。开了它有个后果：金属度高的模型放进没有环境贴图的Three.js场景里会渲染成死黑。这条在 § 12有完整的修法，现在你只要记住，模型进网页之后黑成一块，多半不是模型坏了。



智能网格模式下的设置弹层要简单得多，只有「拓扑」一组：四边面（标New，默认选中）/三角面，加一个500–25000的「面数控制」滑块，默认5000。左栏的AI模型是「P2.0 – 抢先体验Trial x2」，底部按钮「生成模型 ⚡ 1000」。国区站studio.tripo3d.com，2026-09-14。

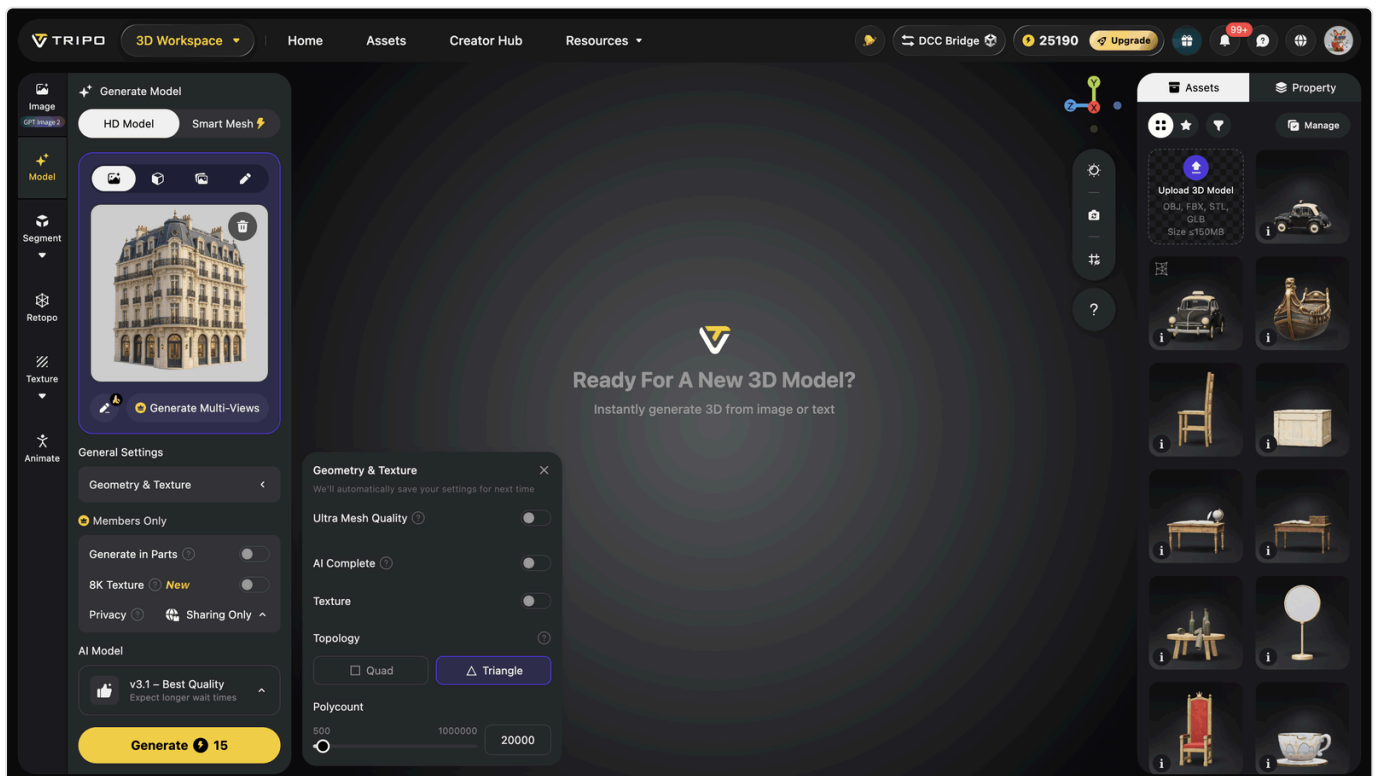
「Generate 50」这个数字是怎么来的

我一开始以为50就是生成一个模型的价钱。后来干脆坐在表单前面做了件很笨的事：把几何与贴图弹层里的开关一个一个动，每动一下看按钮上那个数跳成多少。全程只看不按，没花积分。



起点是默认状态：Ultra Mesh Quality关、Texture开、Texture Quality选中8K、PBR开、Topology Triangle、Polycount 20000，左栏Members Only里8K Texture开着。传的是一张奥斯曼转角公寓设计图，按钮「Generate ⚡ 50」。2026-09-11 14:40。

只动一个开关：把Texture关掉。贴图那几行当场全部收起，Texture Quality、Remove Lighting、PBR一起消失，连左栏的8K Texture也灭了。按钮从50掉到15。



同一张图、同一个表单，唯一的差别是Texture关了。弹层里只剩Ultra Mesh Quality、AI Complete、Topology、Polycount四项，按钮变成「Generate ⚡ 15」。关掉的其实是一整包东西，不止那一个开关。2026-09-11。

再把Texture开回来，逐档试分辨率，得到的是一串很规整的数：

几何与贴图里的状态	按钮上的数	贴图这一包占多少
Texture关（只要几何，出灰模）	15	0
Texture开，Texture Quality 2K	30	+15
Texture开，4K	40	+25
Texture开，8K（默认）	50	+35
8K再打开Ultra Mesh Quality	65	+35，Ultra另算15

结构就很清楚了：几何15是底价，贴图是个打包加项，2K加15、4K加25、8K加35，Ultra Mesh Quality再加15。默认那50分里，35分是8K贴图，几何本身只占15。

这里有个容易搞混的地方：Ultra Mesh Quality那15分不在默认的50里面。工作台打开时Ultra是关着的，50已经是它关着时的价；开了才变65。把Ultra算进默认价，一整批资产的预算会凭空多出四分之一。每个动作的基础价和加项，完整放在 § 07。

还有一件事我是被它坑了才知道的。当天早上我截到过一张按钮写65的表单，还以为是另一种模式。其实是我前一次生成把Ultra Mesh Quality开着，弹层顶上那句「We'll automatically save your settings for next time」替我记住了。这个记忆是跨会话的：你为了一件特写资产开的档，会一直跟着后面每一件。

首页那个快捷框又是另一套默认值，同一张图选高精度模型时按钮写55，比工作台的50多5分。首页那个框没有可展开的设置，所以这5分落在哪个开关上，我在界面上核不到，也就不去猜。能用的规矩只有一条：以你按下去之前按钮上当场显示的那个数为准，两个入口的默认开关不一样。同一张图想要同一个价，就都从工作台进，那边每个开关都看得见。

核心建议

要批量做几十件资产，开工前先花五分钟把这个弹层设一遍：Texture Quality拨到2K，Ultra确认是关的。一件30和一件65，乘以40件差1400积分，接近半个月的订阅额度。

走一遍智能网格：从灰模到上色

下面是我2026-09-11那天用同一张出租车设计图跑的一次完整流程，用的是智能网格的P2.0预览版（免费那一次）。

1 传图

输入区四个页签：图片上传、多视图、文本、画笔。图片支持JPG/PNG/WEBP，20 MB以内。我传的是一张1024×1024、浅灰底、四分之三视角的出租车设计图（怎么准备这张图，见 § 02）。

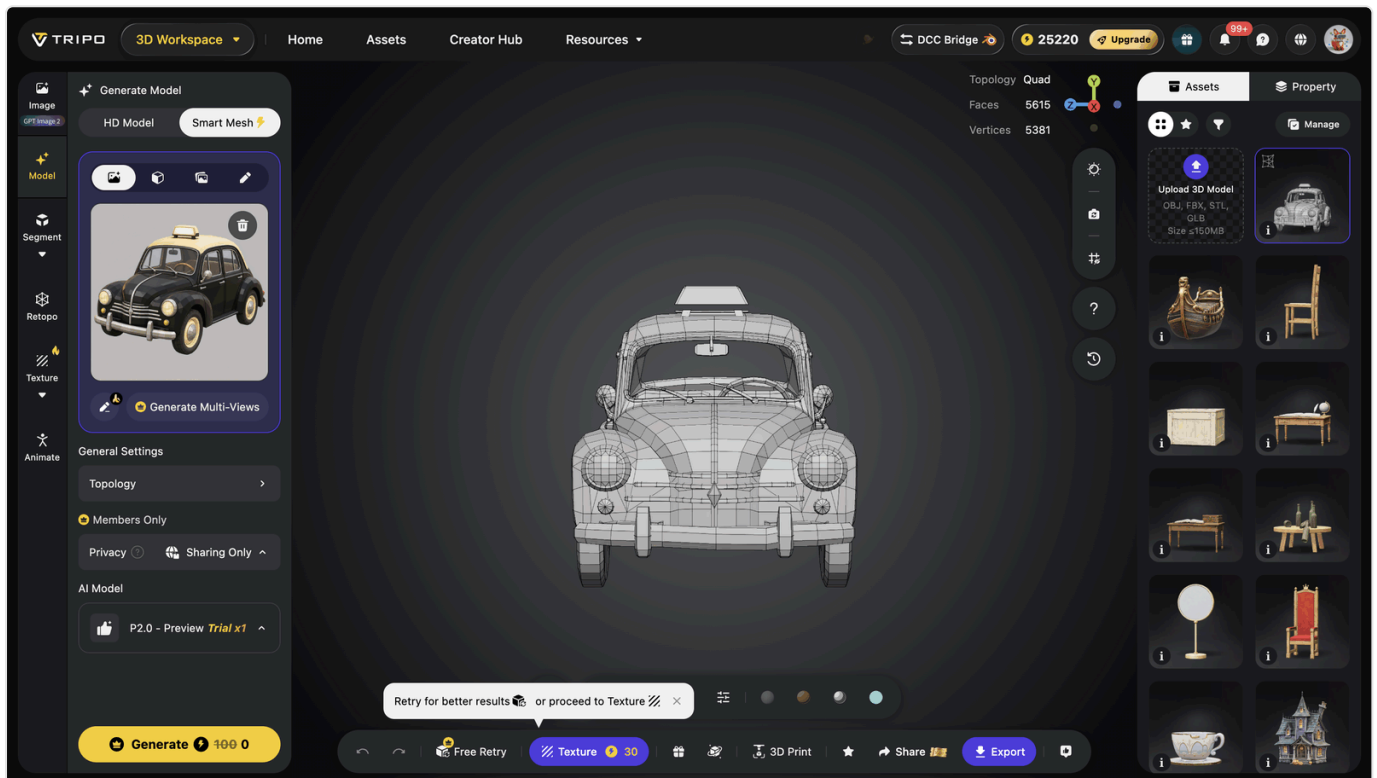
2 选模式和拓扑

智能网格 → AI Model选P2.0 - Preview → 拓扑保持四边面，Polycount保持5000。按钮显示「100划掉，0」，这是预览期那一次免费试用；用掉之后同样这一屏是「100划掉，65」。

3 出灰模

几十秒后中间出来一辆没有颜色的车，右上角读数：Topology Quad，Faces 5615，Vertices 5381。底部弹一句「Retry for better results or proceed to Texture」。

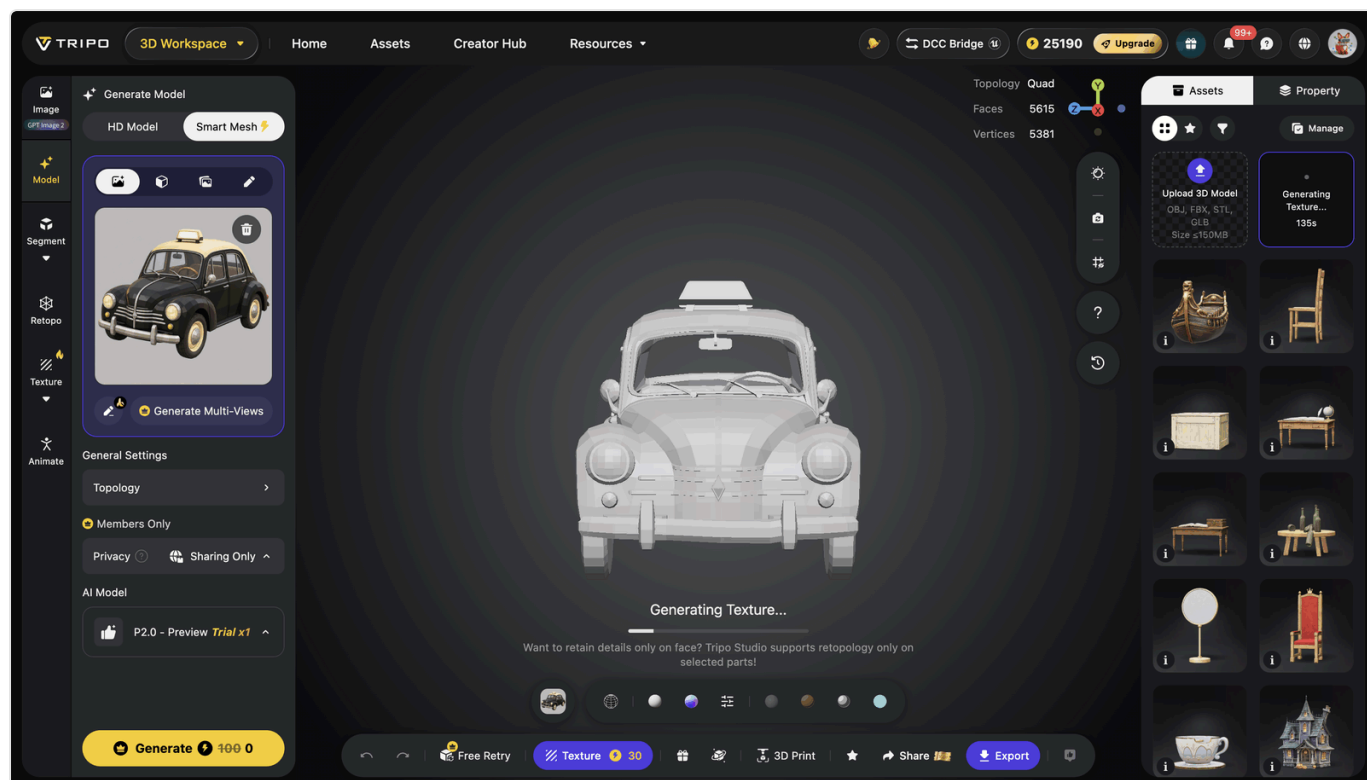
这个「几十秒」要拆两个口径说，见下面「耗时到底该怎么念」。



Smart Mesh（P2.0 Preview）出的第一版：无贴图灰模，四边面的走线在正面看得很清楚。右上角Faces 5615 / Vertices 5381。底栏依次是Free Retry、Texture 30、3D Print、Share、Export。2026-09-11。

4 付30做贴图

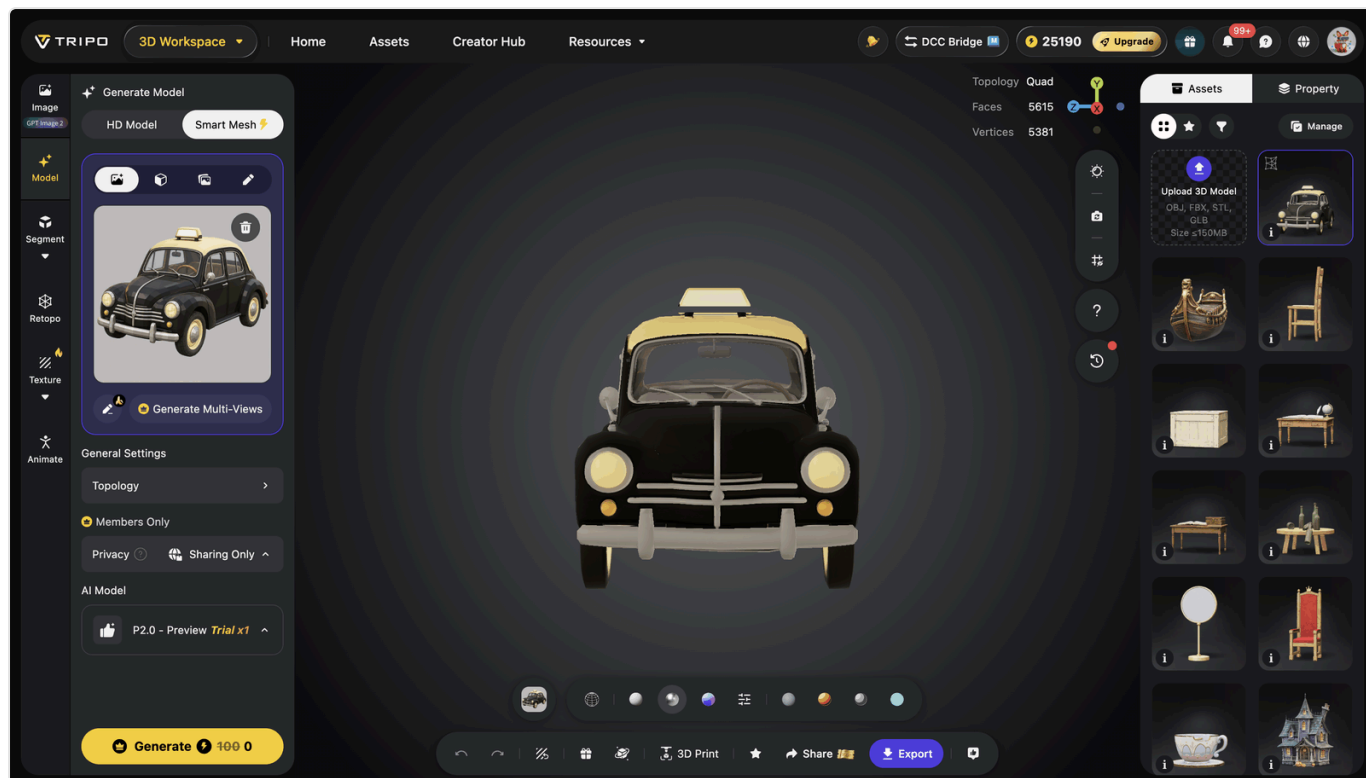
智能网格出的是灰模，要颜色得点底栏那个「Texture ⚡ 30」。点完中间开始跑进度条，标着还剩多少秒。



点了Texture之后的中间状态：模型变成白模，下面「Generating Texture...」进度条，右侧资产卡片上写着135s。卡片上这个135s是界面自己的倒计时，含排队，不是算力耗时——2026-09-12那批里我从接口核过一件贴图生成，算力本身是109秒。顶栏余额这时已经从25220变成25190。2026-09-11。

5 看结果

两分多钟后车身上色完成。底部那一排小球是预览用的环境和材质切换，不改模型本身，只改你现在看它的光。



贴图完成。黑车身、米黄车顶和顶灯、米色保险杠，和参考图一致。底部一排圆球是预览环境 / 材质切换。顶栏余额25190。2026-09-11。

这一趟的账很清楚：生成0（预览期免费），贴图30，余额25220变成25190。试用只有这一次，之后同样一趟是65加30。同一张图我之前用v3.1 HD跑过一次，Studio右上角报的是18497个三角面、12522个顶点；P2.0这版是5615个四边面，三角化之后是9,800个三角面。进引擎的三角面差了将近一倍，但形体没塌，车灯、格栅、保险杠、顶灯该有的都在。

核心建议

18497这个数是界面读数，附录A那张逐件表上同一件写的是18,509面 / 12,770点，不是印错了。两个数来自两条通道：18497 / 12522是Studio右上角报的，18,509 / 12,770是把落盘的 `car-01-taxi.glb` 拿过来直接数索引数组数出来的，和本节后面那批P2.0资产标的是同一种数法。两条通道在这一件上差12个面、248个顶点。要拿附录A当验收单逐件核，以文件读数为准；界面上那个数用来当场判断「要不要减面」够用了。

注意

四边面有一条现在就得知道的连带后果：Tripo的智能拆分（Segment）不吃四边面模型，也不吃已经绑过骨的模型，界面上直接写着「Unavailable for: Quad models, Rigged models」。所以你要是打算把这个模型拆成车身和轮子分别转，那就别在这一步选四边面，得用三角面的那一版去拆。这条在 § 05展开。

耗时到底该怎么念

做完一件我总想知道「这到底几秒」，然后发现这个问题有三个不同的答案，而它们都是对的。下面这一列数来自2026-09-12录屏时现场生成的另一辆出租车（智能网格，Quad，6044面）：

口径	那一件读出来是	怎么量的
算子耗时	16秒	打接口读 data.operator 的 updated_at - created_at ，这是Tripo自己记的算子起止
端到端（含上传与排队）	约31秒	录屏上数帧：点下「生成」到车出现在画面里
界面倒计时	卡片上写多少算多少	它是个估计值，会跳，别拿它当测量

这本书里凡是给了秒数的地方，写的都是算子耗时，不含上传和排队。我把口径写死在这儿，是因为这两个数差着将近一倍，你拿16秒去估自己做四十件要多久，会估少一半。图片上传本身也要时间，你的网络和图的大小都算在里面。

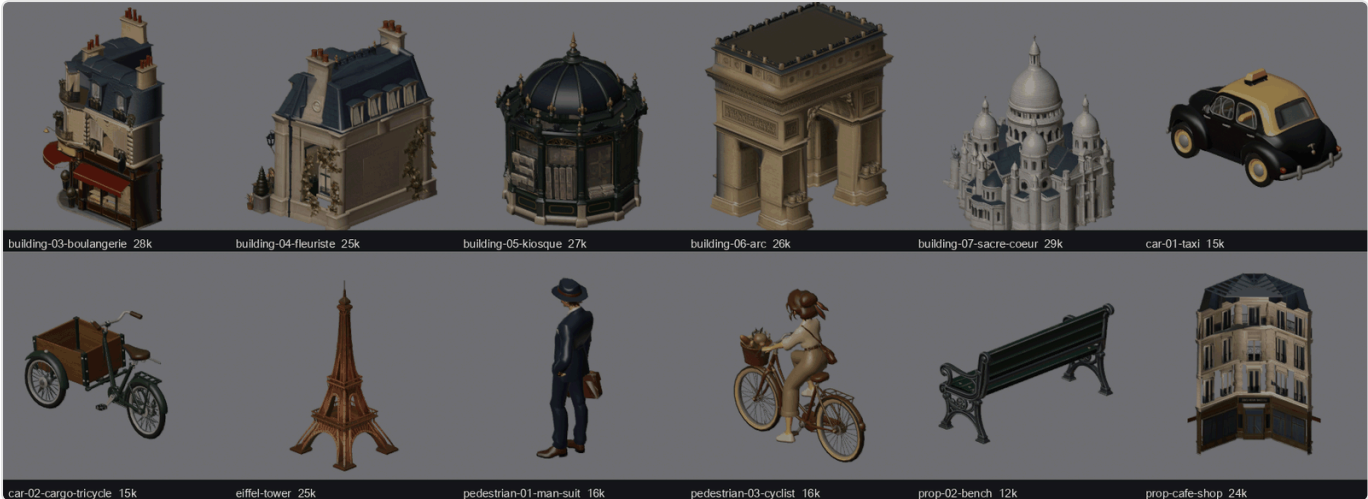
一次批量：十四件网格的面数怎么定

2026-09-12我把巴黎那套资产整批重做了一遍，全部走智能网格。这一批是这本书里最像「真的在干活」的一段：十四件网格、十三件纹理，一个下午。

面数没有一个万能值，我按用途分了三档：

档位	填多少	这一批里的实例	三角化之后实际落在
建筑与地标	15000	凯旋门、圣心堂、面包房、花店、报刊亭、铁塔、街角咖啡馆	24,495–29,587三角面
人物与车辆	8000	出租车、载货三轮、西装男、骑车女孩	15,431–16,126三角面
街道道具	6000	长椅	12,752三角面

看最右边那一列你就明白上一节为什么要专门讲面数口径了：填6000的长椅，进引擎是一万两千多个三角面。一来实际面数会略微超过你填的那个数，二来四边面三角化还要再乘一道，实测倍数在1.37到1.87之间、中位1.75（完整分布见附录D）。做预算请直接拿这一列的三角面数去算，别拿表单里填的那个数去算。



2026-09-12那批智能网格资产的本地渲图总览（12件，另有一件绑骨用的行人不在图内）。每件下方是槽位名和三角化之后的三角面数：面包房28k、花店25k、报刊亭27k、凯旋门26k、圣心堂29k、出租车15k、载货三轮15k、铁塔25k、西装男16k、骑车女孩16k、长椅12k、街角咖啡馆24k。这些数是从最终GLB的索引数组里数出来的，不是界面读数。

耗时也很有参考价值，下面这些全部是算子口径：最快17秒（长椅），最慢65秒（街角咖啡馆），建筑大多落在45到62秒，人物和车辆落在21到26秒。大体的规律是面数填得越高、结构越碎，跑得越久。

十二件的算子状态全是 `success`，一次重试都没用上。这和我做高精度那批的体感一致：图准备对了，生成这一步本身很稳。真正废掉的那一件，问题出在参数没送进去，下面单说。

这一批唯一浪费掉的65分

凯旋门那一件我做废了，废得很隐蔽：我用浏览器自动化往面数框里打字，框里明明白白写着15000，提交完打接口核参数，`face_limit` 读回来是5000——也就是默认值，65积分打了水漂。

手动点鼠标打字的人不会遇到这条。但只要你打算让Codex或者Claude Code替你批量跑表单——这本书的读者大概率会——这就是第一个坑。根因、正确的写法、还有一个不用等生成完就能验的硬办法（设完面数按一下左右方向键看它跳到几），完整在 § 04。

另外三条只有连着做几十件才会遇到的

现象	怎么绕过去
面数设置不被记住。弹层顶上那句「自动保存您的设置」管得了模式和加项，管不了面数，每一件都会回到默认	把「设面数」当成每件必做的一步写进流程，别指望上一件的设置还在
提交完一件接着传下一张图，上传会失败	每提交一件就重新进一次生成页。页面上那个图片输入控件在提交后被前端清掉了，不回去重新来一遍，第二张图根本传不上
刚回到生成页那一下，界面是「降级」状态：价格、时间戳都显示得怪	不用管，把图片传上去它自己就恢复正常了

十分钟清单

把上面浓缩成一张开工前过一遍的单子：

推荐

照这个来

- 先在几何与贴图里把用不上的加项关掉，设一次管后面所有次
- 面数在Polycount滑块上定，别在提示词里求
- 面数按用途分档：建筑15000、人车8000、街具6000
- 做网页和游戏，生成时Texture Quality选2K，导出那一步选1k
- 打算拆件的，拓扑留三角面
- 按生成按钮之前先看一眼上面那个数字

不推荐

别这样

- 一路默认点过去，五十分五十分地花，其中三十五分买的是你导出时又要降掉的8K贴图
- 拿我这个两万多的余额去估自己的成本
- 8K贴图原样丢进网页，然后怀疑是Three.js的问题
- 用四边面模型去做智能拆分，卡在那儿找不到原因
- 拿两种模式的面数直接比大小
- 以为智能网格那65里包含贴图（贴图另付30）

到这里你手上应该有一个能转的模型了。下一节讲那张送进去的图该长什么样——**图生3D的成败，八成不在这些开关上，在你喂进去的那张图上。**

§02 参考图怎么准备

Preparing the Reference Image

图生3D的成败八成在那张图上。这一节给你一条能背下来的四字配方和它的完整提示词原文、五条展开的硬要求、一个当场判断成没成的验收信号，外加两个我真的翻过的车。

四条配方，先抄走

整本书如果你只带走三样东西，这是其中一样。参考图的配方，压到能背下来的长度就四条：

核心建议

参考图四条配方

- ①一个物体
- ②白底
- ③四分之三视角
- ④整个东西都在画面里，别裁

四条都满足，图生3D的命中率会高到让你觉得这一步不需要技巧。缺一条就开始有东西不对，缺两条基本要重来。

四条翻译成一句能直接丢给出图模型的话，是这样，**原样抄走就能用**：

单个物体、完整入画、居中放在纯白背景上，四分之三视角，
只有极淡的接触阴影、无其他物体、无文字、无人物。

这句话我在所有设计图的提示词里都挂着，一个字没改过。下面把四条拆开讲为什么，顺带补上第五条——它不在配方里，我却在它身上摔过一跤。

配方之外，这一节剩下的事都在后面两节展开，这里不重复：**五条硬要求各挡什么、一整套能出风格一致批次的提示词模板、换题材怎么改、两次翻车（梵高的柏树变成岩石尖塔、活塞裁掉一截变成硬币）、以及开工前的一分钟检查，全在 § 09；为什么先出设计图而不是文生3D、以及「读Tripo给它起的名字」这个验收信号，在 § 08。**

模板里那些「无XX」只是倾向，不保证

模板前半段里写着「无阴影无文字水印」。这句话大部分时候管用，偶尔不管用。



2026-09-12补做的街角咖啡馆参考图，1024×1024。提示词里明写了「无文字水印」，模型还是在雨棚上写了一行「LE CHAT NOIR CAFE」。这张图我照样送进去生成了，出来的模型没问题——文字被当成贴图上的纹样烘进去了，没有变成形体。

遇到这种情况先别急着重画。先分清一件事：那个没听进去的东西会不会变成形体。

没听进去的是	要不要重画
雨棚上多了一行店名	不用。平贴在表面的图案会进贴图，不进几何
脚下多了一片投影	看深浅。极淡的接触阴影没事，带方向的长影子可能被当成一块地面板
画面里多了第二样东西	重画。这一条最贵，它会真的被造出来，粘在主体上

所以四条配方里真正不能让步的是「一个物体」，另外三条都还有一点弹性。提示词没被完全执行，本身不构成重画的理由。

正面例子：一张横幅怎么改成能用的输入

做旋转游乐场那个demo需要两匹旋转木马。参考图找的是美国Index of American Design的一张水彩记录画，维基图页上标的是公有领域，横幅构图。（同上，这两件木马也只在随包素材这一层被剔掉了，口径见 S 00。）



原始图，维基图页标注为公有领域：一匹白色旋转木马，蓝鬃、黄红马鞍、绿色坐垫，侧身四分之三、四条腿腾空、完整入画。构图是横幅，左右大量留白。



处理后送进Tripo的版本：没有裁，是把横幅补成1024×1024的正方形，主体比例一点没动。背景的米白纸面统一保留。

关键动作是补背景，不是裁主体。原图本来就满足五条要求里的四条：单物体、全身入画、浅色背景、接近四分之三视角。缺的只是比例，那就补背景，别动主体。

结果是两匹木马都一次过，Tripo认成ornate horse和carousel horse，各50积分。后来做旋转逻辑的时候还发现一条：Tripo出的木马朝向是固定的，长轴沿Z，代码里统一转九十度就能让它们沿切线站好。

§03 生成之后

After the Generate Button

模型跑出来了，接下来三件事：看懂右上角那三行读数、在正确的地方验收、把文件真的拿到手上。最后一件不止「点导出」一条路，做到十件以上，走接口比点按钮省事得多。

右上角那三行数

生成完成，画面右上角会出现三行：Topology、Faces、Vertices。这是你对这个模型的第一手了解，比看图靠谱。

生成完成后右上角那块读数区见 § 01那张图：Topology Quad、Faces 5615、Vertices 5381；底栏从左到右是撤销 / 重做、Free Retry、Texture 30、3D Print、收藏、Share、Export。

读数	怎么读
Topology	Quad是四边面，Triangle是三角面。这一行决定了它后面能不能拆件（四边面不能），也决定了拿进Blender好不好改。
Faces	面数。这是它在你的游戏或者网页里占多少性能的第一指标。
Vertices	顶点数。和面数一起看，能大致判断网格干不干净。

巴黎那批资产的面数，我有一份完整统计可以当参考：31个GLB、29个独立资产，面数从16711到46196，**中位数18509，一个超过10万面的都没有。**

这份统计里有一件得单独拎出来说，不然它会骗你：**46196那个最大值，是奥斯曼建筑重拓扑之后的面数，它直出时是92.7万面。**那是镂空阳台栏杆那种结构，复杂度由对象本身带来——同一批里结构简单的埃菲尔铁塔，直出只有1.9万面。这一件也是29件里唯一走过重拓扑的，**其余28件全是直出，一次重拓扑都没用上**，默认就落在1.5到2万面这个游戏可用的区间里。那92.7万面怎么压到2万多的四边面拓扑、代价是什么，在 § 04；同一件在减面实验里的逐行读数在 § 17。

想让面数从一开始就落在你要的量级，办法在按下生成之前：高精度模型的「几何与贴图」弹层最底下有个Polycount滑块，在那儿先把数定死就行（§ 04）。我做奥斯曼建筑那一件的时候还没翻到这一栏——§ 04开头那句被我自己推翻的错话，就是这么来的；后面几批一律先设面数，再没出现过这个量级。

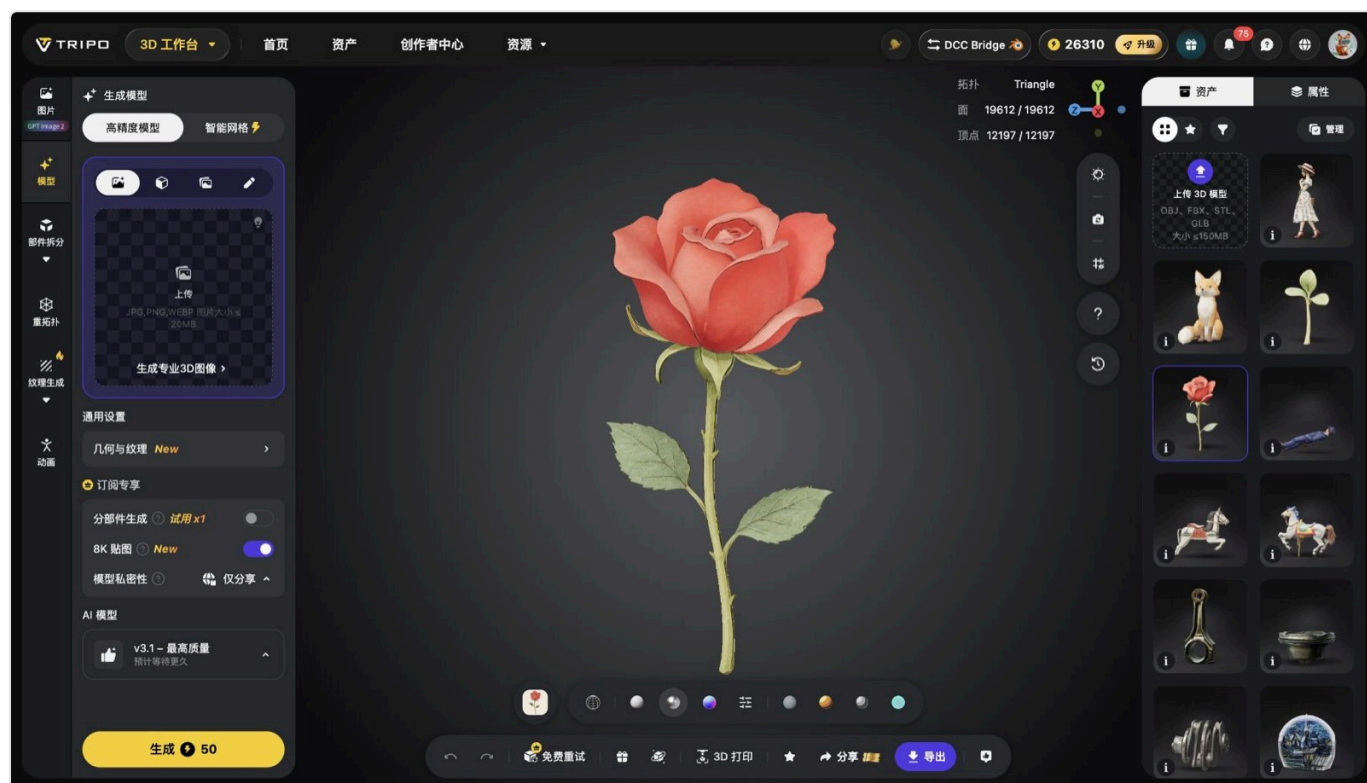
面数预算：一个场景总共能承受多少个面。这个说法本身带一个陷阱：**它诱导你先定一个绝对数再往里塞资产**——我原先就给过「30万三角面以内」这么一个数，后来把书里真跑过的读数排到一起，发现它站不住，而且错的方向是让你砍过头，把本来塞得下的东西砍掉。正确的次序是先量GPU耗时和绘制调用，量不出问题就去动面数（§ 04）。

验收去Tripo自己的预览页，不在你的引擎里

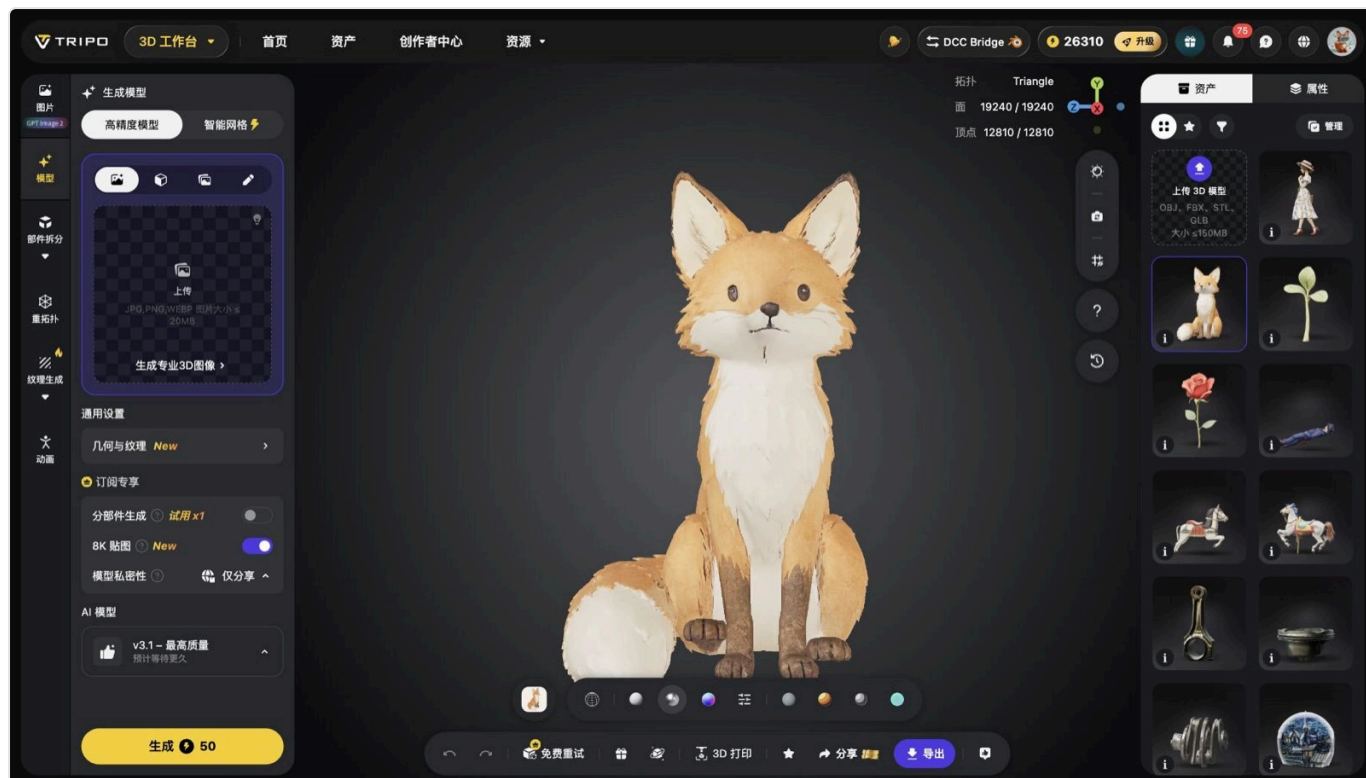
这一条是我踩坑踩出来的，而且踩得很典型。

做小王子那个demo的时候，我一度判定玫瑰和狐狸两个模型都重建失败了。在我自己的引擎里看，玫瑰是「一团皱巴巴的红块」，狐狸是「摊在地上的一坨」。我当时甚至已经往经验库里写了一条归纳：细长结构是图生3D的弱项。

两次判断都是错的，那条归纳也是错的。



回Tripo预览页看同一个玫瑰：花头、花萼、细长花茎、两片叶子全部正确，茎上连刺都有。右上角读数：Triangle，面19612，顶点12197。左侧v3.1最高质量、8K贴图开。界面是中文版，2026-09-10。



狐狸也一样：坐姿、尖耳朵、黑鼻子、白胸口、蓬松尾巴，一样不缺。面19240，顶点12810。2026-09-10。

错在哪儿？我是在夜景、低角度、贴得很近的截图里下的判断。那种条件下，低多边形模型会退化成一个剪影，红玫瑰缩成一个红团，狐狸缩成一坨。这是我判定方式的问题，不是资产的问题。

更要命的是，Tripo的预览页是免费的、直接的、权威的一手证据，任务详情页一进去就是它。我却跳过它，拿自己引擎里一张光都没打对的截图下了结论。

核心建议

正确的验收姿势：先看Tripo预览页。要在自己的场景里看，就把条件摆正：中性光、正面、相机退到能看见整体。还有一条：真正的比例和落点只能从包围盒和离地间隙里读，不能从截图里估。

做到十几件之后，预览页就不够用了

一件一件点开预览页，点到一半就不想再点了。2026-09-12那批十三件，我改成把文件取到本地（怎么取，下面两节就讲），用固定机位批量渲一组图，一次看完。

```
/Applications/Blender.app/Contents/MacOS/Blender --background --factory-startup \
--python _verify/render_glb.py -- <某个.glb> <输出.png> 640
```

它出四张，机位按glTF的坐标约定固定死：-Z（正前）、+X（右）、+Y（顶）、斜45°。全程不开浏览器、不点鼠标，十三件一起跑几分钟就完。



其中一件（巴黎西装男行人）的四视图，2026-09-12本地渲染。从左到右依次是glTF的-Z、+X、+Y、斜45°机位。-Z那一张他是侧身的，+X那一张才正对镜头——这就是「这件模型的正面朝哪」的答案，摆进场景之前必须先知道。

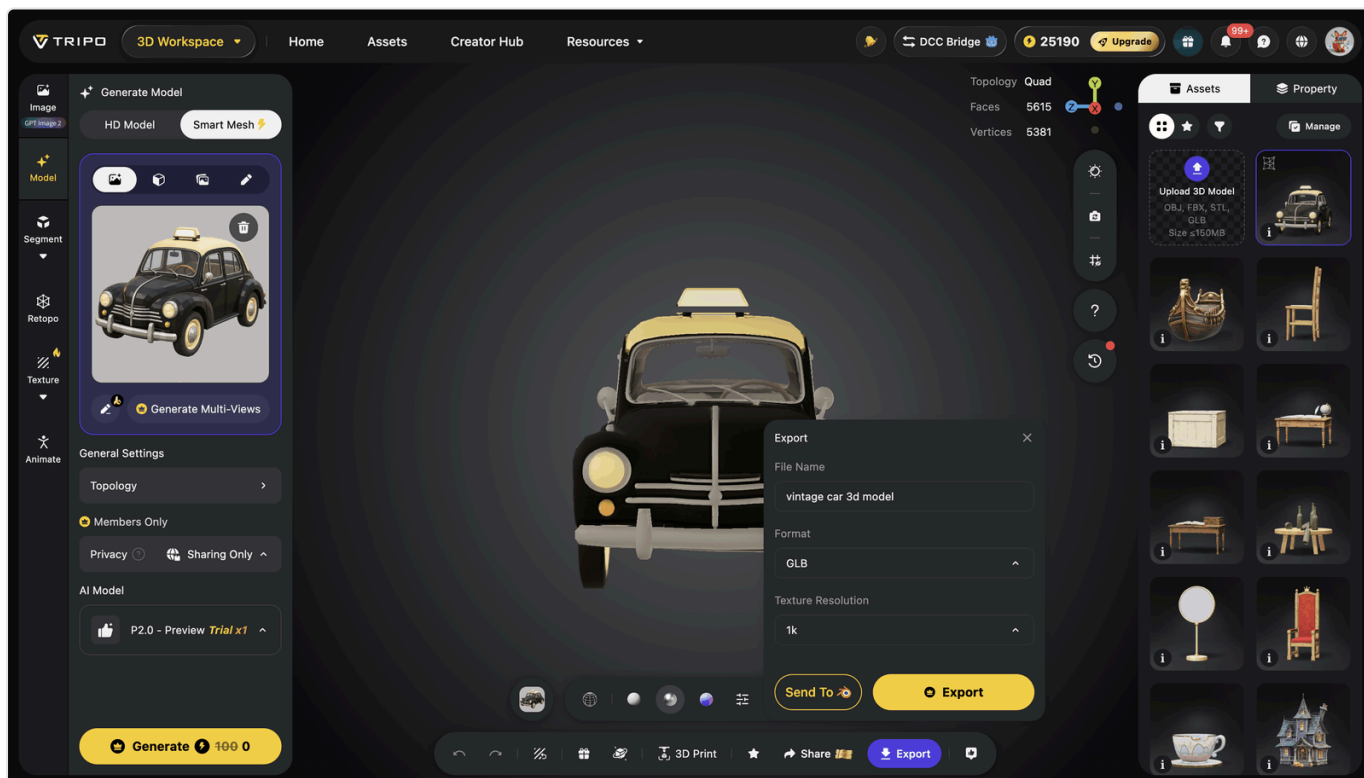
这组图能一眼看出三件事：贴图有没有贴歪、比例对不对、**正面朝哪个方向**。最后那条最容易被忽略，代价也最大——等一整条街的行人都背对着马路才发现，要返工的是摆放代码，不是模型。各家资产的朝向口径不一样，§ 12有完整的一张表。

注意

四视图只有一个斜机位，正面和背面在缩略图里很容易看混，建筑尤其如此——四个立面长得都差不多。要把朝向彻底定死，就把机位加到六个（前后左右顶斜）再渲一遍，几分钟的本地渲染而已。

导出弹层

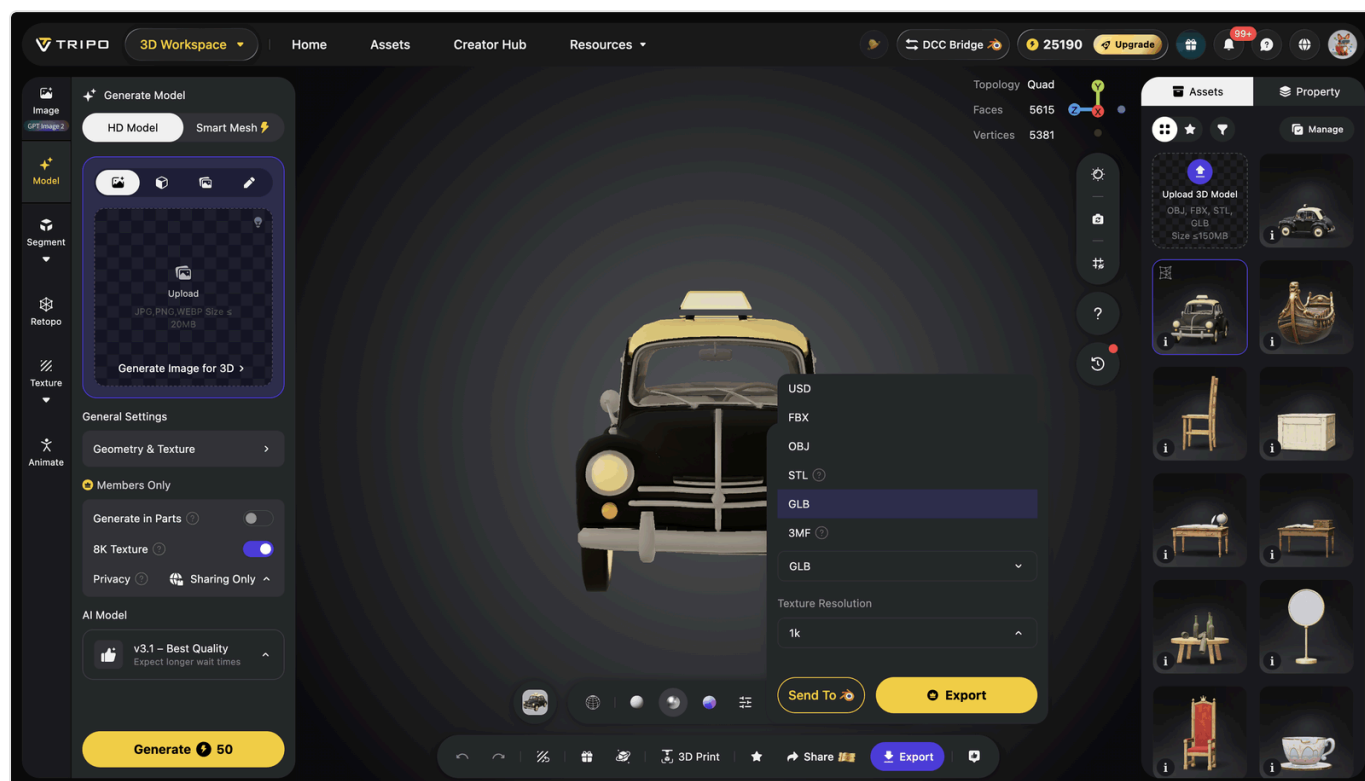
底栏那个蓝色「Export」按钮点开，是一个四行的弹层。



导出弹层。File Name那一栏预填的「vintage car 3d model」就是Tripo给这个模型起的自动名，Format选GLB，Texture Resolution选1k，左下角「Send To」旁边那个图标是直接推到Blender的DCC Bridge。2026-09-11。

File Name: 预填的就是 § 02 说的那个自动命名。这辆车它认成了「vintage car 3d model」，对的。你要改名字在这儿改，但改之前先读一遍原来那个。

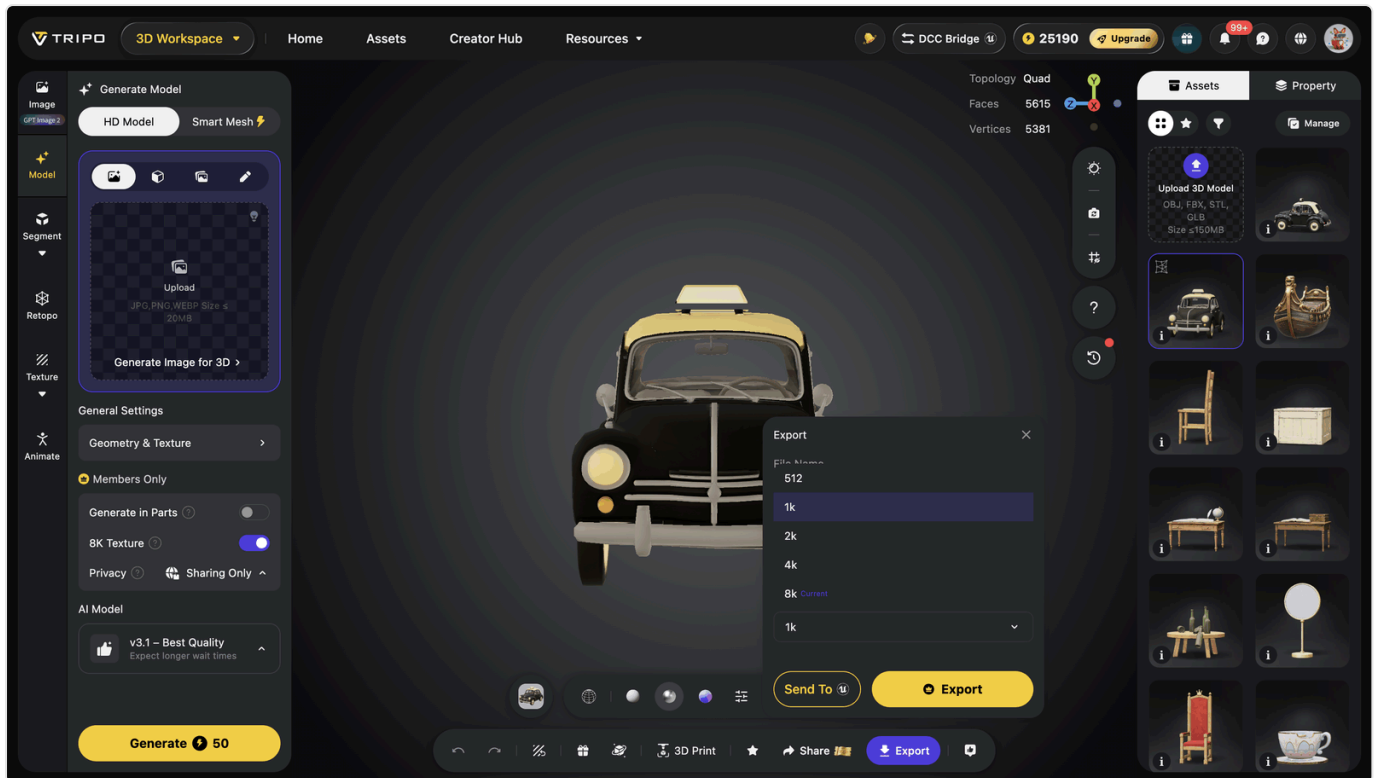
Format: 下拉展开是六种，USD、FBX、OBJ、STL、GLB、3MF，默认落在GLB上。



Format下拉展开：USD / FBX / OBJ / STL / GLB / 3MF，高亮那一项GLB是默认值。STL和3MF后面各带一个问号，那两个是3D打印用的。模型是Smart Mesh P2.0那辆出租车，右上角还挂着Quad / 5615 / 5381。2026-09-11。

我这次全程只导GLB，网页和引擎都吃它，一个文件里把网格和贴图全打包了。其余五种我没有逐个导出验过，所以只说界面给了什么，不说它们各自好不好用。上传那一侧接受的是OBJ、FBX、STL、GLB四种（工作台右侧「上传3D模型」那个框上写着，150 MB以内），比导出少一个USD和一个3MF。

Texture Resolution: 这里选1k。这是 § 01 那条警告的落点。Tripo默认导8192×8192，一张底图摊进显存是358 MB，十几件资产就能吃掉5.0 GiB显存。在这个下拉里选1k，同样的场景降到0.22 GiB，游戏距离上肉眼看不出区别（这两个GiB是显卡工具那侧按1024进位的读数，换成本书算显存的十进制口径是5.38 GB和0.24 GB，口径见 § 12）。



Texture Resolution下拉：512 / 1k / 2k / 4k / 8k五档，8k后面标着蓝色的Current，意思是这个模型身上现在挂的就是8k。高亮的1k是弹层打开时的默认选项。整个弹层里没有任何一处标着积分数，导出这一步不额外收费，所以贴图在这儿降档是白拿的。2026-09-11。

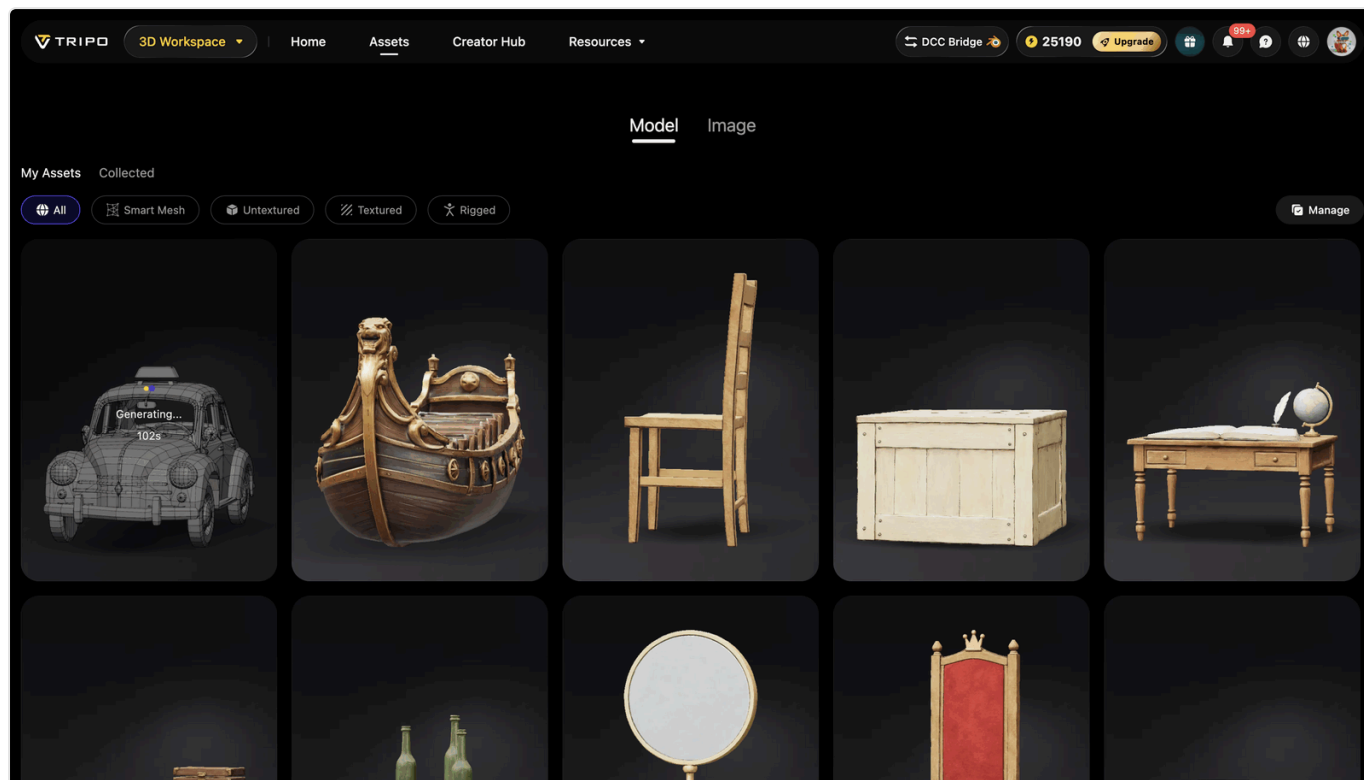
「不收费」这句话我后来补了一次正面证据。做那组模式对照实验的时候，我从详情页导过一次FBX，导之前余额20205，导完还是20205，一分不动。导出格式转换这一步确实不吃积分，你要GLB要FBX要OBJ，想导几次导几次。

Send To：直接推给DCC Bridge，不落本地文件。要接Blender的话这是最短的路，具体在 § 11。

注意

关于导出还有一条得先说清楚，我手上两个来源不一样强，都摆在这儿。官方Compare Plans对照表里，Free那一列Exports的原文是15 (H2.5 only)——额度是有的，只是被限定在H2.5这一档上。另一边，我的项目记录里写着免费账号生成得出来、但GLB / FBX这些格式全部导不出、必须订阅，我自己是从专业版起步的，这一条没在免费账号上复现过，强度不如那张官方表。全书这件事的统一口径在 § 07，对照表原文和逐行读数在附录C。你要是想先免费试试水，在花钱之前先把「导出」这一步、连同你真正要的那个格式一起点到底，别等做了一堆资产才发现拿不走。

Assets页：你的东西都在这儿



Assets页（顶栏「Assets」进）。上面是Model / Image两个页签，下面My Assets / Collected，再下面一排筛选：All、Smart Mesh、Untextured、Textured、Rigged。左上角那辆车正在跑，卡片上写着还剩102s——这是界面自己的倒计时，含排队，和§01说的算子耗时是两个口径。2026-09-11。

那排筛选条比看起来有用。资产做到几十件的时候，「哪些还没上贴图」「哪些已经绑过骨」靠肉眼在缩略图里翻是翻不出来的，点一下Untextured或者Rigged就齐了。

走接口拿GLB：绕开那个「另存为」对话框

点「导出」走的是浏览器下载链路，会弹一个系统级的「另存为」对话框。你自己点没问题，但只要你想批量、想让AI agent替你跑，到那个对话框就得停手——原生文件对话框是任何浏览器自动化都进不去的地方。

所以我换了条路：不碰那个按钮，直接去走数据本身的通路。按该用的顺序有三条。

路子一：官方开放API（正式集成走这条）

要把取文件这件事写进脚本、接进自己的工具、交给agent自动跑，正经入口是官方开放API：基址 `https://openapi.tripo3d.com/v3`，API Key在`platform.tripo3d.com`创建，认证走 `Authorization: Bearer <token>`，任务轮到 `status: "success"` 之后从 `output.model_url` 取GLB。这条路是公开文档写明的，字段有版本号兜着，不会因为网页改版而失效。

有一个前提得先知道：**API的额度和Studio的订阅是两套账，要分别购买**。所以走这条路之前先去 `platform.tripo3d.com/billing` 把API钱包充上，别拿Studio的余额去推算它。完整的接法、CLI和插件怎么用，在§11。

路子二：应急——只有几件、又暂时没配API Key时

先说清楚定位：签名链接约两天就过期，这条通路也没有版本承诺，正式集成请走路子一。它只解决一种情况——手上就三五件，API Key还没申请下来，先把文件拿到手。

页面加载模型预览的时候，本来就会去请求CDN上那个GLB。在资产列表里点一下，让详情页把模型真的加载出来，打开浏览器的网络面板，在请求列表里找那条 .glb ，右键复制地址就行。

那是一条CloudFront签名URL，过期了回页面重新加载一次，换一条新的出来。

路子三：知道站内接口存在就行，别依赖它

Studio网页自身还有一套站内接口，2026-09-12那批资产我当时就是从那儿取的。但它没有公开文档、没有版本承诺，路径和字段随时会变，还得挂着你浏览器里的登录态，所以我不把具体怎么调写进来。要批量、要写进产品、要交给agent长期跑，请走路子一的开放API——那才是有版本号兜着的那条。

不管URL是从路子一还是路子二拿到的，下载那一步我写了个小脚本，一并做解码和校验：

```
python3 pull_glb.py --url '<编码过的URL>' --out 产出/rose.glb
# URL 太长或者带特殊字符，就先存成文件再喂：
python3 pull_glb.py --url-file /tmp/u.txt --out 产出/rose.glb
```

它做三件事：URL如果被编码过就解一次码，下载，然后校验文件头四个字节是不是 gLTF 。最后打印文件名、体积和哈希前十六位。这一步的意义是拿到正面证据：拿到的确实是一个GLB，不是一个改了扩展名的错误页。

注意

签名URL里带着 & 和 ~ ，千万别在shell里手敲或者直接粘进命令行，这两个字符会被shell吃掉，下载回来是个坏文件而且不一定报错。要么交给Python的requests，要么把整串原样写进文件再读。

还有一条容易踩的：Smart Mesh出的模型，取到手的成品文件是 output_mesh_<id>.fbx ，不是GLB。别拿GLB的解析器去读它然后怀疑文件坏了。带贴图的那一版是 tripo_texture_<id>.fbx ，绑骨那两个是 tripo_rigging_<id>.fbx 和 tripo_retargert_<id>.fbx ，都是FBX。要进网页得先转成GLB，那道工序在 § 17。

十三件串下来是什么体感

不走导出按钮批量取文件这件事，我在2026-09-12那批上跑满了一轮，有几个数值得先知道：

项	实际
单件体积	8-16 MB（里面95%是那张8192×8192的贴图，几何只占一点点）
单件下载	1-2分钟
十三件串行跑完	约20分钟。连绑骨那两个文件一共十五个，150 MB

项	实际
失败	一次。building-05-kiosque 撞上 SSL: UNEXPECTED_EOF_WHILE_READING，重跑就好

核心建议

批量下载的脚本一定要写成可以重复跑的：文件已经在本地就跳过，只补缺的那几件。十三件跑二十分钟，中断一次，如果没有这个跳过逻辑，你就得从头再来一遍二十分钟。这一条比任何下载技巧都值钱。

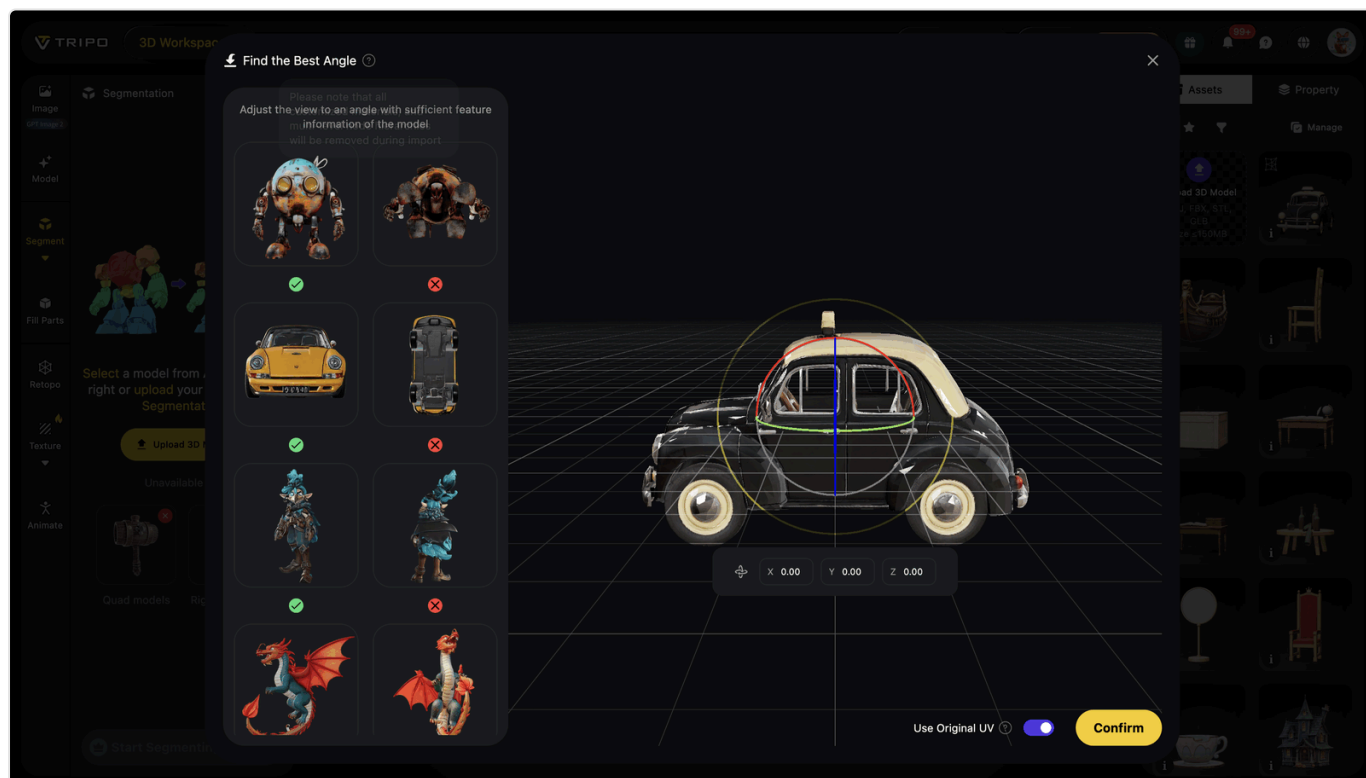
把模型传回Tripo接着做

导出不等于结束。减面、拆件、绑骨这些都在Tripo里做，你也可以把外面的模型传进来。上传支持OBJ、FBX、STL、GLB，150 MB以内。

传进去之后会先弹一个对话框，要你把模型转到一个「特征信息足够的角度」。

注意

这个弹层只在上传外部模型时出现。如果那件模型本来就在你自己的Tripo账号里（比如刚在工作台生成的），从页面右侧的资产栏直接点选就行：不弹这个对话框，不用转角度，也不收那笔50积分的导入费。上传这条路是给外部文件准备的——Blender里改过的、别的工具出的、别人给你的。



上传模型后的「Find the Best Angle」对话框。左侧那一列是官方给的四组对照示例，每组一个打勾一个打叉：机器人正面四分之三打勾、背面打叉；黄色跑车正面打勾、底面打叉；角色正面打勾、背面打叉；最后一组是只生物，侧身打勾、正对打叉。右边把模型拖到合适角度，左下角有个「Use Original UV」开关，确认后开始。2026-09-11。

看左边那一列示例，你会发现它和 § 02 讲参考图的四条配方是同一个道理：能看见特征的那一面才算信息，看不见的那一面等于没给。参考图是这样，模型的分析角度也是这样。

核心建议

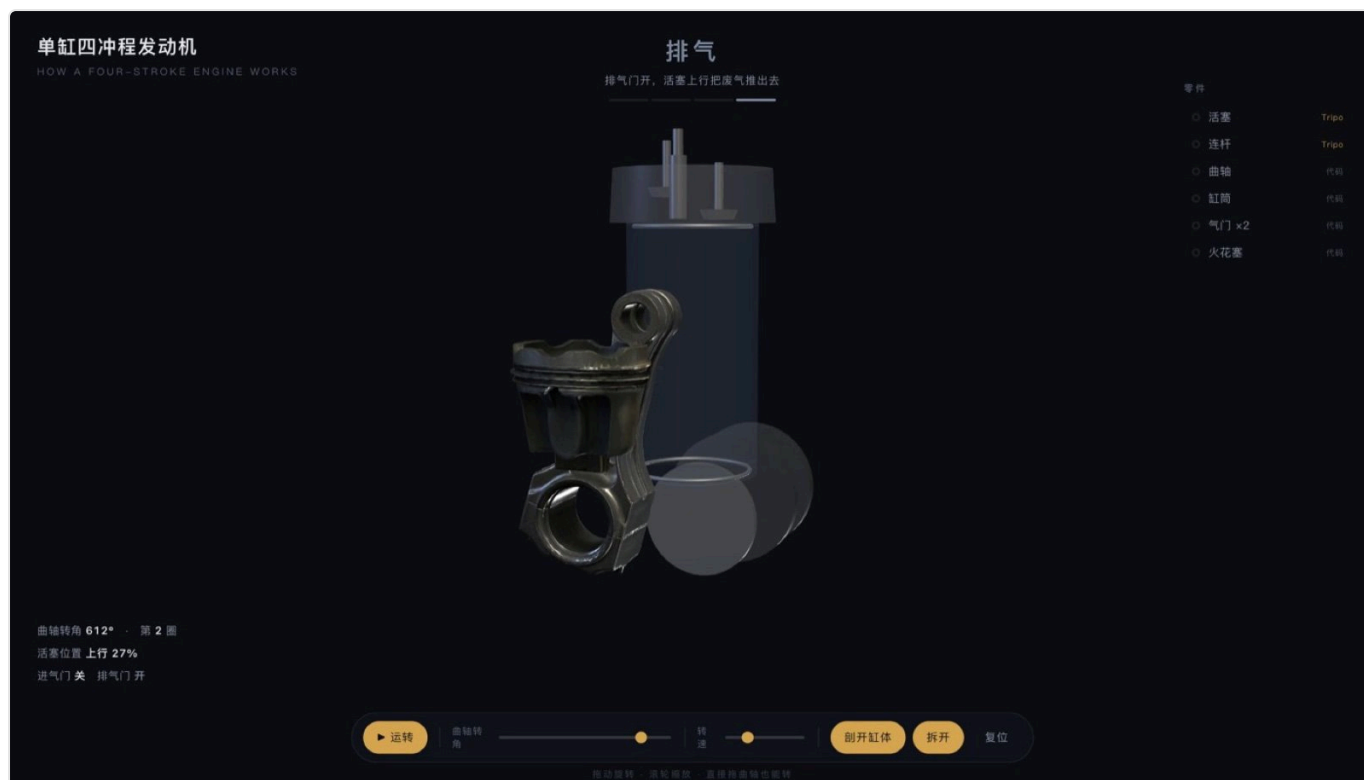
由此有一条贯穿全书的默认顺序：**能留在Tripo账号里做完的事，就别先导出再传回去**。生成、拆件、重拓扑、绑骨、纹理都在同一个账号里接着做，最后一次性把成品取走。**导出是终点，不是中转站**。这条在绑骨那一步差价最明显，§ 06 有一整段算这笔账。

拿到文件之后去哪

GLB到手，后面的路就宽了。我这两周把它们送去了三个方向：Three.js做成能在浏览器里走进进去的世界、Unity做成原生可玩的版本、Blender做减面和绑定的返工。



Tripo导出的GLB接进Three.js之后：小王子站在直径28米的B612上，角色、玫瑰、猴面包树、水井这些都是Tripo生成的资产，星球表面是程序化的。完整做法在 § 22。



另一个方向：四冲程发动机的可拆解demo，白底、爆炸视图。活塞和连杆是Tripo生成的，曲轴、缸筒、缸盖是代码画的。为什么这么分工，在 § 15和 § 20。

```

# 重新跑一次 P2.0 的签名直链下载（落到临时目录，不覆盖已归档的产物）
OK taxi-p2-smartmesh-textured.fbx 7,473,036 bytes head=b'Kaydara '
OK taxi-v31-segmented.glb 6,606,904 bytes head=b'glTF\x02\x00\x00\x00'
OK taxi-segmentation-map.png 619,828 bytes head=b'\x89PNG\r\n\x1a\n'
python3 0.17s user 0.17s system 0% cpu 34.483 total
alchain@MacBook-Pro-127 _pull-run %

```

下载脚本跑完的真实终端输出：三个文件各自的字节数与文件头都打出来了（FBX的 Kaydara、GLB的 glTF、PNG的 \x89PNG）。这是第二次跑的结果。第一次那条6.3 MB的GLB报了 The read operation timed out ——不是签名过期（有效期还剩两天），就是读超时。「拿到正面证据」说的正是这件事：文件名对、大小对都不算数，要看文件头。2026-09-11。

这一节要记住的

推荐

记住这几条

- 结果出来先读Topology / Faces / Vertices，再读自动命名
- 验收在Tripo预览页做，别在自己没打好光的场景里判死刑
- 十件以上就改成本地渲固定机位，顺手把朝向定下来
- 导出时贴图选1k；要批量别点导出按钮，正式集成走官方 openapi.tripo3d.com/v3
- 批量下载的脚本写成能重复跑的，断了只补缺的那几件

不推荐

别这么干

- 在夜景里看一眼就判定模型失败
- 8K贴图原样导出，然后去优化渲染代码
- 把签名URL粘进命令行手敲
- 账号里已经有的模型，导出来再传回去（那是一笔50积分的导入费）
- 免费账号做了一堆资产才去试导出

到这里，「一张图变成一个能用的模型文件」这条最短路径就走完了。接下来两节讲怎么把它变便宜、变轻、变成可以拆开动起来的零件。

§04 减面与重拓扑

Decimation and Retopology

读完这节，你会知道自己手上这个模型到底要不要减面、面数该定多少、什么时候花那10积分走一次重拓扑。中间夹着一场我花15积分做的受控对照实验：同一张图、同一个面数上限，只换几何模式那一栏，两边出来的网格根本不是一回事。还有一个更重要的事实：拖慢你游戏的多半不是三角形。

先去数一遍面数，多半你用不上重拓扑

我做巴黎那个可驾驶街区demo的时候，心里一直装着一句自己写下的话：网页版默认把面数拉满，prompt里写low poly也不听，所以必须走一次重拓扑二次处理。这句话在项目README里躺了两天，后来我把31个GLB文件全解析了一遍，才发现它是错的。

真实数字是这样：**29个独立资产，面数落在16711到46196之间，中位数18509，没有一个超过10万面。**（31个文件里pedestrian-01有静态、绑骨、动画三个版本，是同一个网格，去重后是29个。）这个量级本来就是游戏能直接用的，整个巴黎项目一次重拓扑都没用上。

那句错话是从哪来的？它来自更早一次实测，那批资产里镂空精细的奥斯曼建筑天生927K面，也就是92.7万，得走重拓扑压到2万多面的四边面拓扑才进得了引擎。我当时在README里把它写成了「927万面」，同一节的下一行写的却是「927K面」，同一个927，一处万一处千，差十倍。自己项目的文档也是待核实来源，不是事实。

所以这一节的第一个动作不是打开重拓扑页，是去数面数。 Tripo Studio工作台右上角常驻三行读数：Topology (Quad还是Triangle)、Faces、Vertices。先看这三个数，再决定要不要花那10积分。

三个词先说清楚

面数就是这个模型由多少个小平面拼成，面越多形状越细腻，显卡每帧要处理的东西也越多。**三角面**是三个顶点围成的面，引擎最终认的都是它；**四边面 (Quad)** 是四个顶点围成的面，做动画和手工改模型时它更听话，因为边能连成一圈一圈的走向。

重拓扑是把同一个形状用一套全新的、更规整的面重新铺一遍，形还是那个形，铺法换了。**减面**（业内也叫decimation、simplify）是在原来那套面上做合并删减，不重铺。两者都能把面数降下来，但重拓扑给你的是干净的边环走向，减面给你的是更少的三角形，代价是拓扑更乱。

上面这三段是定义。定义谁都能背，问题是四边面到底值多少钱、值不值你多付的那几十积分。这件事我不想靠转述别人的说法，所以专门跑了一次对照实验。

E10：同一张图，只换模式那一栏

这个实验编号E10，2026-09-12跑的，总共花掉15积分。要回答的问题只有一句：**同一张参考图，走高精度模型和走智能网格，出来的网格差在哪，差多少。**网上能查到的说法全是定性的——「四边面更好改」「三角面更适合直接渲染」——我想要能数出来的数。

第一步：把输入锁死

对照实验最容易垮在「两边其实不一样」上。所以先说控住了什么。

	A · 高精度模型 (v3.1)	B · 智能网格 (P2.0)
项目uuid	17263734-46ba...	d8d0edde-a37f...
提交时间	09-12 15:01	09-12 14:03
参考图	car-01-taxi.png	同一张
face_limit	8000	8000
texture	false	false
拓扑	三角面	四边面

「同一张图」这四个字我不想靠记忆。把两个项目的接口返回里 `image_to_model.image` 指向的 `input.png` 各自拉回本地，连同源文件一起算SHA-256，三方对：

A 的 input.png	1,141,563 字节	c2c01442...f102b6ab
B 的 input.png	1,141,563 字节	c2c01442...f102b6ab
本地 car-01-taxi.png		c2c01442...f102b6ab

逐字节相同。有意思的是这两次上传在Tripo那边是两个不同的对象（`20260912/0f78f762.../input.png` 和 `20260912/fd811639.../input.png`），路径不一样、内容同一张。**哈希这一步别省。**「我记得我传的是同一张」和「这两个文件的SHA-256逐位相同」，在证据等级上差着一整个数量级，而一个对照实验的全部价值就押在这个等级上。

注意

「两边参数完全一样」这句话不能说，得说准。表单上我动的只有模式那一栏，但拓扑那一项是跟着模式走的：智能网格这边四边面是默认选中，高精度那边我选的是三角面。所以严格讲，变的是「模式+它自带的拓扑」这一对，不是一个孤立字段。这是个产品行为细节，写实验报告的时候必须交代，不然后面那个2.09倍的结论就成了含混账。

另外还有一个我根本控不住的变量：图生3D会给你的图自动配一句英文描述，两次跑出来措辞不一样（A是 `vintage car with black body, beige roof...`，B是 `rounded vintage car with black body and cream roof...`）。这一句是模型看图现写的，表单里根本没有它的位置，每次都会飘。

第二步：数出来的结果

	A · 高精度模型v3.1	B · 智能网格P2.0
四边形	0 (0%)	6,835 (79.51%)
三角形	7,379 (100%)	1,761 (20.49%)
N边形	0	0
面数 (原生)	7,379	8,596
面数 (三角化后)	7,379	15,431
顶点	3,646	8,580
生成耗时	62秒	本轮没取到，见下
积分	15	65
model_version	v3.1-20260211	Nexus-v2.0-20260801
输出格式	tripo_base_model_<id>.meshopt.glb	output_mesh_<id>.fbx
包围盒 (归一化后长宽高)	1.000 × 0.533 × 0.567	1.000 × 0.511 × 0.538

那个62秒是**算子耗时**，从接口的1789196500减到1789196562算出来的（算子耗时口径，不含上传与排队；端到端要往上加，两个口径的实测差见 § 01，词条见附录D）。

核心建议

「积分」那一行是两种模式各自的单价，15加65这个80不是这场实验的支出——节首说的15才是。B那件不是为这个实验单独跑的：它就是09-12主批12件网格里的第4件（当天14:03提交），那65积分早算进那批「12 × 65 = 780」里了，逐笔怎么和当天余额对上的在 § 07。A是同一天15:01才单独跑的，**15积分是E10这一轮唯一新掏的钱**。拿一件现成的来做对照，前提是参考图和 `face_limit` 都对得上，这两样上面已经用哈希和接口读数核过。

第三步：三条独立通道互相核对

四边形占比这个数有个麻烦：**只有解原生文件才看得到**。平台预览会三角化，导成GLB也会三角化，你在网页里和在Three.js里都数不出来它原本是几边形。所以这个数必须自己去文件里挖，而只要是自己挖的数，就得有第二条路来验它。这次用了三条。

1 自己写的FBX解析器

FBX把多边形存在 `PolygonVertexIndex` 这个数组里，每个多边形的最后一个顶点索引会被写成负数当结束标记。顺着数组走，遇到负数就切一刀，切出来那一段有几个数，这个面就是几边形。脚本是

`e10_topology.py`，解析器复用的是9月11日那次P2.0实测写的 `_probe_fbx.py`。

2 Blender 4.4的 `mesh.polygons`

把同一个FBX导进Blender，遍历 `mesh.polygons`，按 `len(poly.vertices)` 分桶再数一遍。这条路和上面那条没有任何共用代码，两边唯一的共同点是读的是同一个文件。

3 Tripo详情页右上角那三行

A那件在Studio里读回来是「拓扑Triangle /面7379 /顶点3646」，和前两条完全吻合。这一条最省事，也是你自己复现时第一个该看的地方。

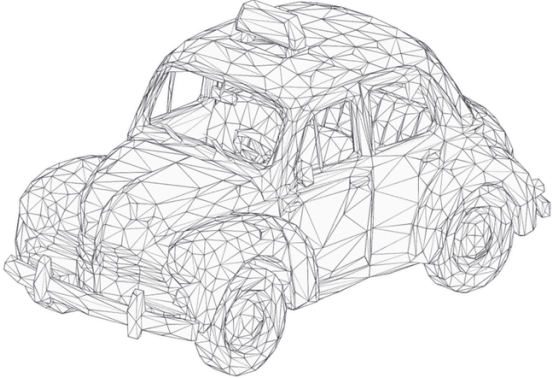
前两条**逐位相同**：四边形、三角形、N边形三个桶的计数一个不差，连「接近」这种说法都用不上。第三条独立确认了A。这样这张表上的数就不是某一个脚本的自说自话了。

注意

有一格我没测到，如实说。B那件的平台读数这次没取到：那件带着8K贴图，在后台标签页里模型加载不出来，预览区是空的。所以B这一列只有两条本地通道，没有平台侧的第三方确认。另外**B的生成耗时我也不敢用**——详情接口只返回项目当前最新的那个算子，B后来跑过贴图生成，现在读回来的时间戳是纹理那一轮的，网格阶段的已经取不回来了。宁可这一格空着，也不去抄一个来源不明的数。

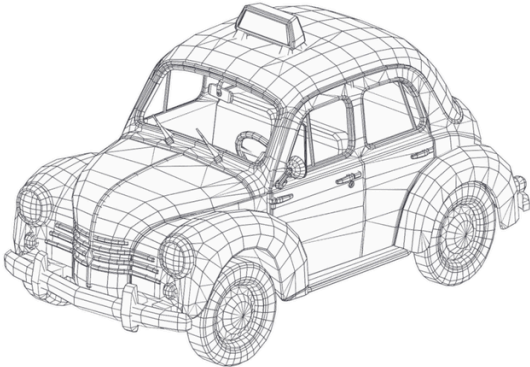
第四步：线框图上一眼可见

同一张参考图 car-01-taxi.png (SHA-256 c2c0144...), face_limit 均为 8000, 只换几何模式



A · 高精度模型 v3.1 (三角网格)

三角形 7,379 (100%) · 四边形 0 · 顶点 3,646 · 15 积分 · 62 秒



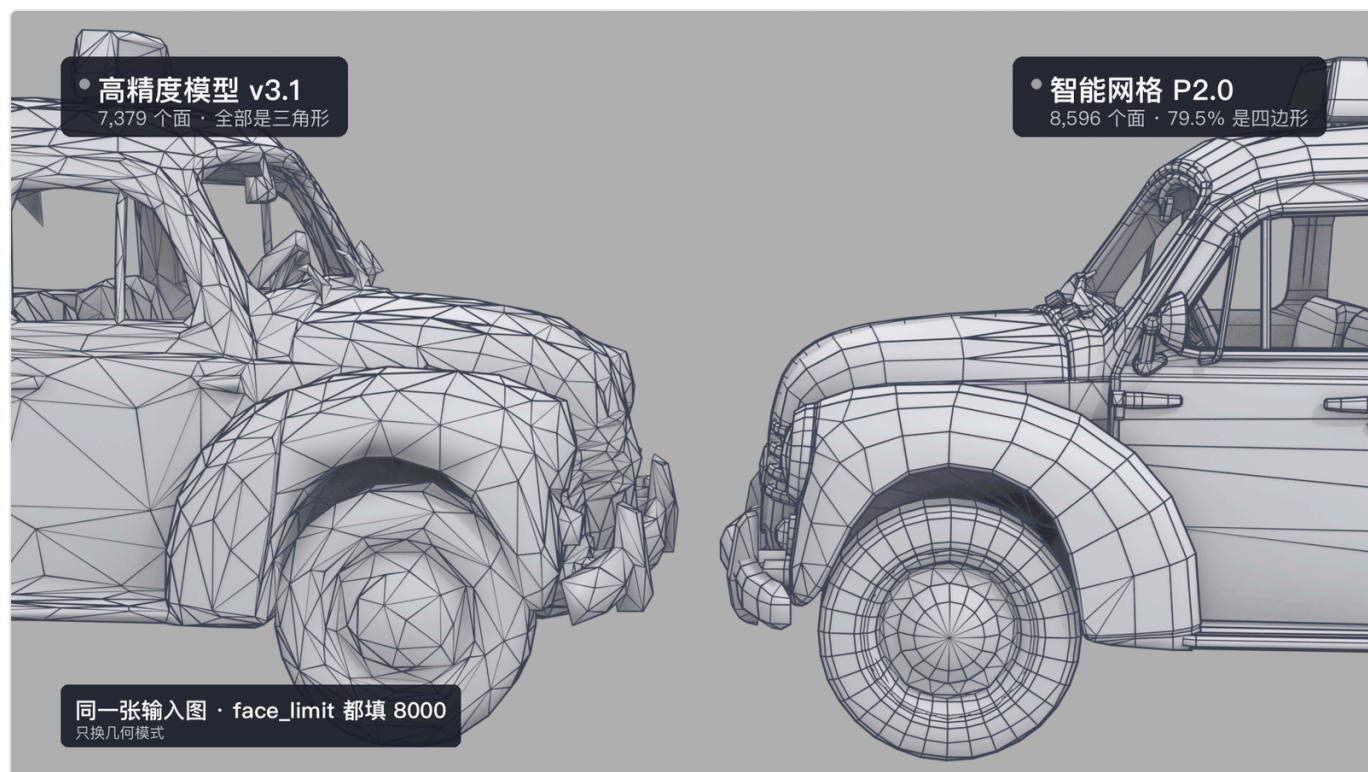
B · 智能网格 P2.0 (Smart Mesh)

四边形 6,835 (79.5%) · 三角形 1,761 · 顶点 8,580 · 65 积分 · 耗时未复核

取证: A 经本地 FBX 解析器、Blender 4.4 mesh.polygons、Tripo 详情页三条通道一致; B 仅本地两条通道一致, 平台侧读数与网格阶段耗时本轮未取到, 故留空。

E10对照线框图 (2084×1236原图)。两张都是白模实体加黑线框、同一个正交45°机位、同一套包围盒归一化。左**A · 高精度模型 v3.1**: 三角形7,379 (100%)、四边形0、顶点3,646、15积分、62秒。右**B · 智能网格P2.0**: 四边形6,835 (79.5%)、三角形1,761、顶点8,580、65积分; 生成耗时那一格图上写的是「未复核」, 不是一个数字——上一版这里印过一个从旧台账抄来的秒数, 本轮接口取不回网格阶段的时间戳, 就把它撤了。图底那行脚注写的是两侧各自的取证通道, 这样图被单独截出去传播时, 证据等级的差别也跟得过去。Blender 4.4.3 headless渲出, 2026-09-12。

左边是一锅**三角汤**: 车身、车顶、轮毂上全是走向随机的三角形, 没有任何一条能辨认的结构线。右边是规则的四边形网格: 车身板件上横平竖直, 轮毂是放射状的四边形, 车窗、车门缝、保险杠周围能看出明确的**边环** (edge loop, 一圈首尾相接、走向一致的边)。



同一组模型推进到车头。左边高精度那版7,379个面全是三角形，轮眉和大灯周围是一片碎三角；右边智能网格8,596个面里79.5%是四边形，轮胎的同心圆、保险杠的横条、门把手都保持着规则走向。注意左边那圈轮胎：同样的位置，A只是一个多边形近似的圆，B是一圈一圈能选中的环。Blender 4.4.3 Eevee Next渲染，1920×1080，2026-09-12。

结论：同一个8000，几何预算差2.09倍

我视频脚本里原本有一句「同一张图，两种模式出来的东西外观差不多，但后面能改的程度完全不一样」。跑完这一轮，前半句得改。

两件的包围盒比例几乎一样（ $1.000 \times 0.533 \times 0.567$ 对 $1.000 \times 0.511 \times 0.538$ ），整体形体确实是同一辆车。但细节层级差得很明显：B有保险杠的栅条、大灯的圈、车门把手、雨刷、车内的方向盘，A在同样的位置是几块糊过去的三角面。

原因在于 **face_limit** 这个数字在两种模式下数的不是同一种东西：

推荐

智能网格的8000数的是**四边面** → 实得8,596个面，三角化之后**15,431**

不推荐

高精度模型的8000数的是**三角形** → 实得7,379个三角面，三角化之后还是7,379

同一个数字填进去，智能网格给你的几何预算是高精度的2.09倍。这条对你排面数预算是有直接后果的：你以为两边填一样的数拿到的是一样大的模型，实际上差一倍还多。真要比外观，得把高精度那边的 **face_limit** 提到16000左右，把三角面配平到15,431附近再比。

所以这一轮的实话是：智能网格在这次对照里**既更细、又更好改**，代价是65积分对15积分。贵的那一档贵在哪，这张表把它摊开了。要纯静态道具、扔进场景里远远看一眼就完事的，15积分那档完全够；要进Blender接着改、要绑骨做动画、要在近处被人看仔细的，多付那50分买的是下面这两个动作。

四边形到底买到了什么：Blender里的两个动作

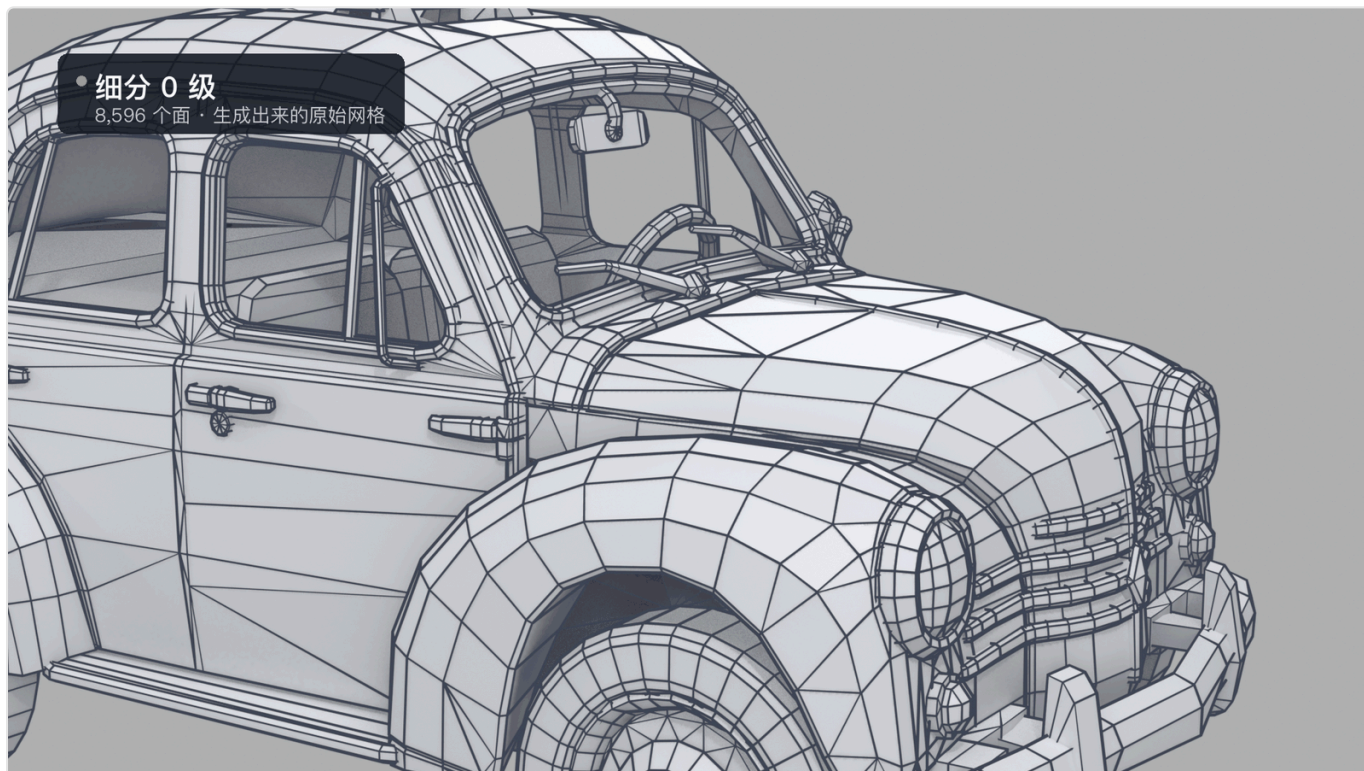
「可编辑性更好」这句话太滑了，谁都能说，听完还是不知道好在哪。所以我把B那件拖进Blender，做了两个最常见的动作，录成了画面。

核心建议

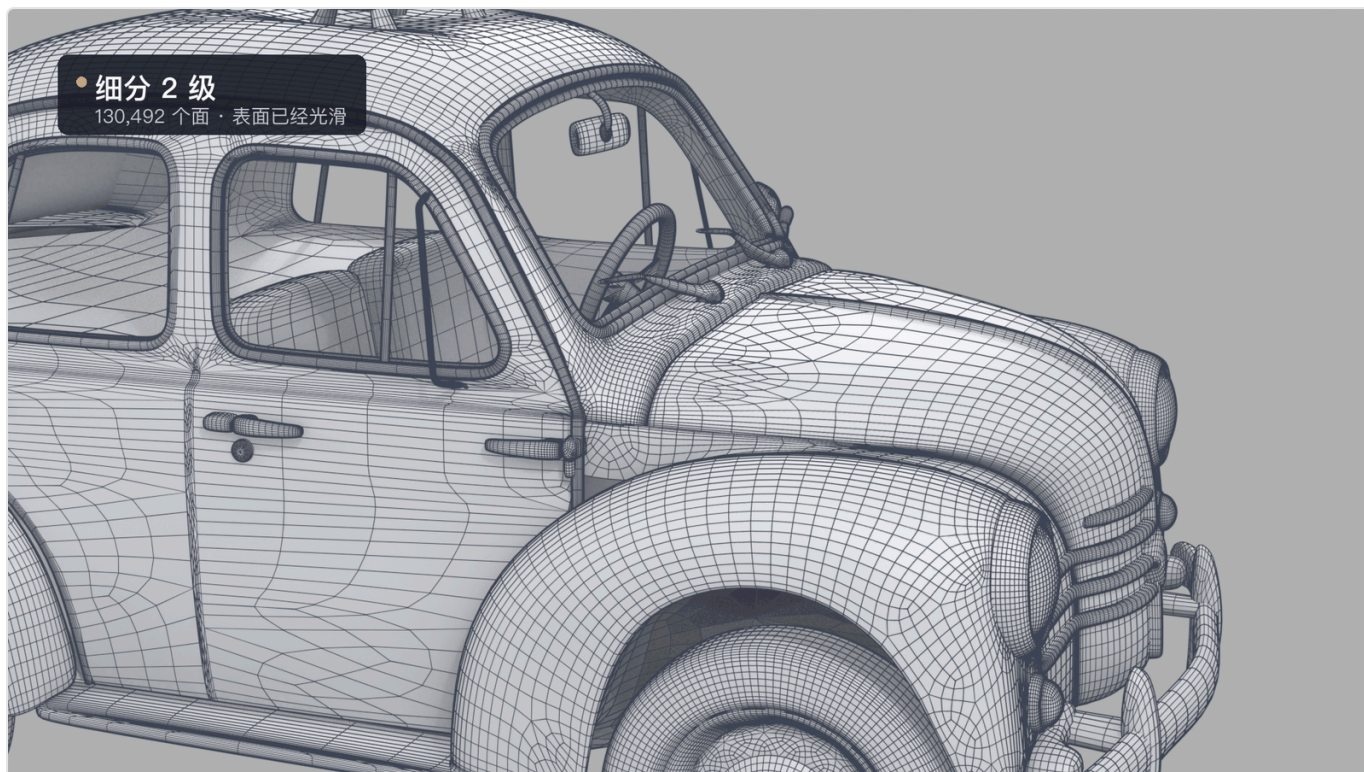
下面这几张是Blender 4.4.3用 `--background` 加EVEE Next离线渲染出来的，1920×1080，全程没开录屏。线框是用Wireframe修改器把每条边变成一根实心管做出来的真几何，不是视口叠加层。好处是画面里没有鼠标、没有菜单栏、没有系统边框；代价是你看不到「Blender长什么样」。要看界面的话得自己开前台录屏。

动作一：推细分

细分（Subdivision）就是把每个面再切细，顺便把表面推光滑。对四边形网格来说这件事特别听话：一个四边形切成四个四边形，网格走向不变，只是密了一倍。



细分0级：8,596个面，也就是Tripo直出的原始网格。车门、引擎盖、轮眉上能看到明显的折面，每一块四边形都是平的。Blender 4.4.3 EEVEE Next渲染，2026-09-12。



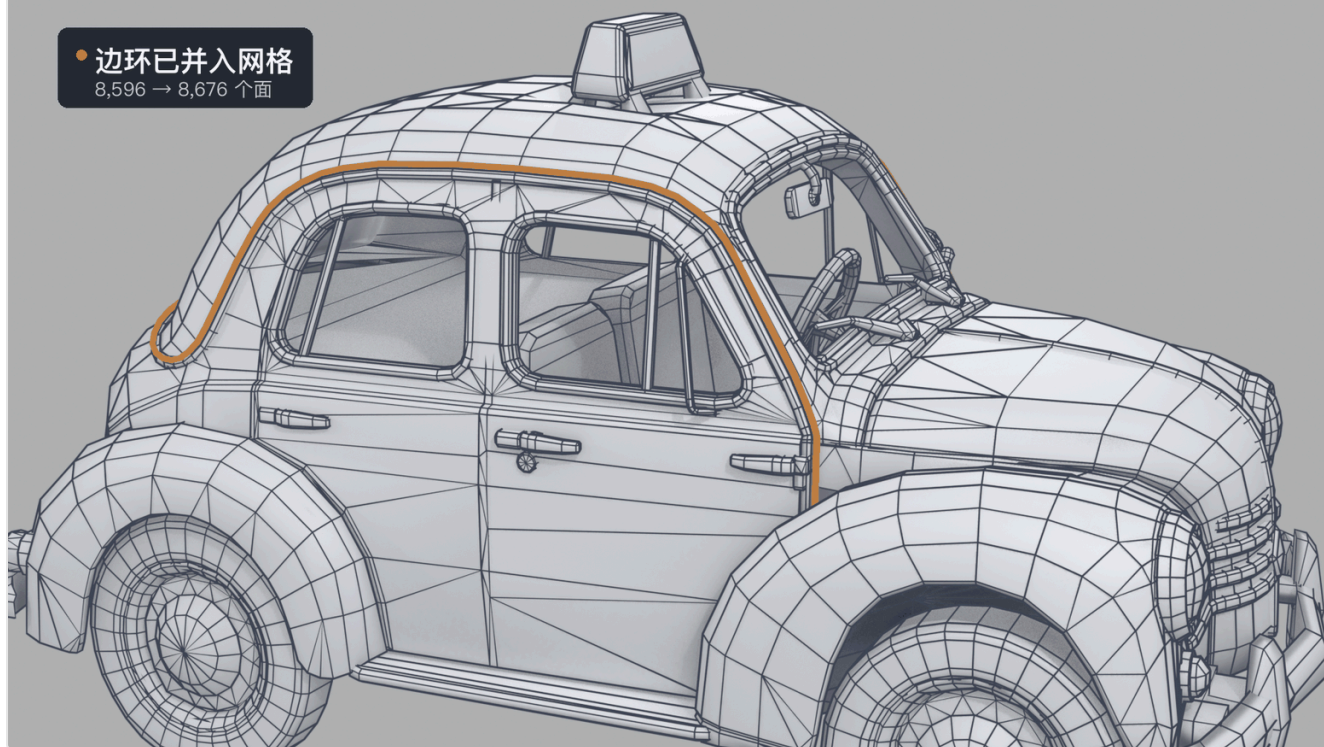
同一个机位、同一个形，细分2级：130,492个面。表面已经完全光滑，门把手、雨刷、大灯边缘的圆角都长了出来。中间那一级没单独放图，细分1级是32,623个面。Blender 4.4.3 EEVEE Next渲染，2026-09-12。

这三个数是能用笔算验的，我很喜欢这一点。Catmull-Clark细分的规则是：一个四边形变四个，一个三角形变三个。B这件有6,835个四边形和1,761个三角形，那么一级之后应该是 $6835 \times 4 + 1761 \times 3 = 32623$ 。和Blender里读出来的32,623一模一样。一级之后全是四边形了，所以二级就是 $32623 \times 4 = 130492$ ，也对上了。能对上这件事本身就是个交叉验证：它同时证明了「6,835个四边形+1,761个三角形」这个拆分是对的。如果四边形占比数错了，这两个数就对不上。

动作二：画一条边环

边环是一整条首尾相连、走向一致的边。在四边形网格上选中它只要点一下，然后就能整条推拉、整条加细、或者在它旁边再切一刀加出新的结构线。这是手工改模型时最常用的动作，没有之一。

● 边环已并入网格
8,596 → 8,676 个面



沿车顶边缘一路绕到A柱的那条边环（81条边），橙色是高亮状态，落刀之后收成一条暗琥珀色的细线，真的切进了网格里：8,596 → 8,676个面，多出来的80个面就是这一圈。这条线是网格的一部分，不是画上去的装饰。Blender 4.4.3，2026-09-12。

有一条得说清楚：这条边环不是闭合的一整圈。它从车后窗一路走到车门上沿就断了。我去查过断点，两端的终止边都只连着一个面，也就是模型在那两处本来就有开口边界，边环走到边界自然停下——不是被三角面卡住的。要形容它，说「一整条边环」比说「一整圈」准。

另外这件模型的拓扑里找不到一条又长又闭合、还正好朝着镜头的竖向横截面环。我扫了全模型1,732条边环去挑，唯一符合的那条只有20条边，而且大半被车身挡住。79.5%四边形不等于「拓扑跟手工做的一样漂亮」，它给你的是「大部分地方能用边环工具」，不是「处处都有教科书级的边环布局」。

注意

细分和边环这两个动作，对三角面的容忍度完全不同，别并排说成「都需要四边形」。按Blender官方手册：Subdivision Surface (Catmull-Clark) 修改器能处理三角面，三角面细分之后会变成三个四边形，虽然会在中心留下一个五星极点、光滑度不如纯四边形网格，但操作本身是成立的——上面那三个能笔算对上的数（8,596 → 32,623 → 130,492）走的就是它；Loop Cut则不同，边环沿着四边形一路往前走，一旦撞上三角面或者N边形，环就在那里终止。所以真正被三角汤卡死的是边环这一类操作。写成「四边形才能细分」是不准的，写成「三角汤上没法拉出一条能用的边环」才是准的。顺带分清一件容易混的事：编辑模式里那个名字很像的Subdivide算是另一个东西，它只把选中的面按边中点切开，默认不做Catmull-Clark那套推光滑，上面那三个数跟它无关。

面数在生成表单里就能定，不用等生成完再补救

这是我自己走了弯路才明白的一条。高精度模型那一侧，展开「General Settings → Geometry & Texture」，弹层最底下就是Topology和Polycount。

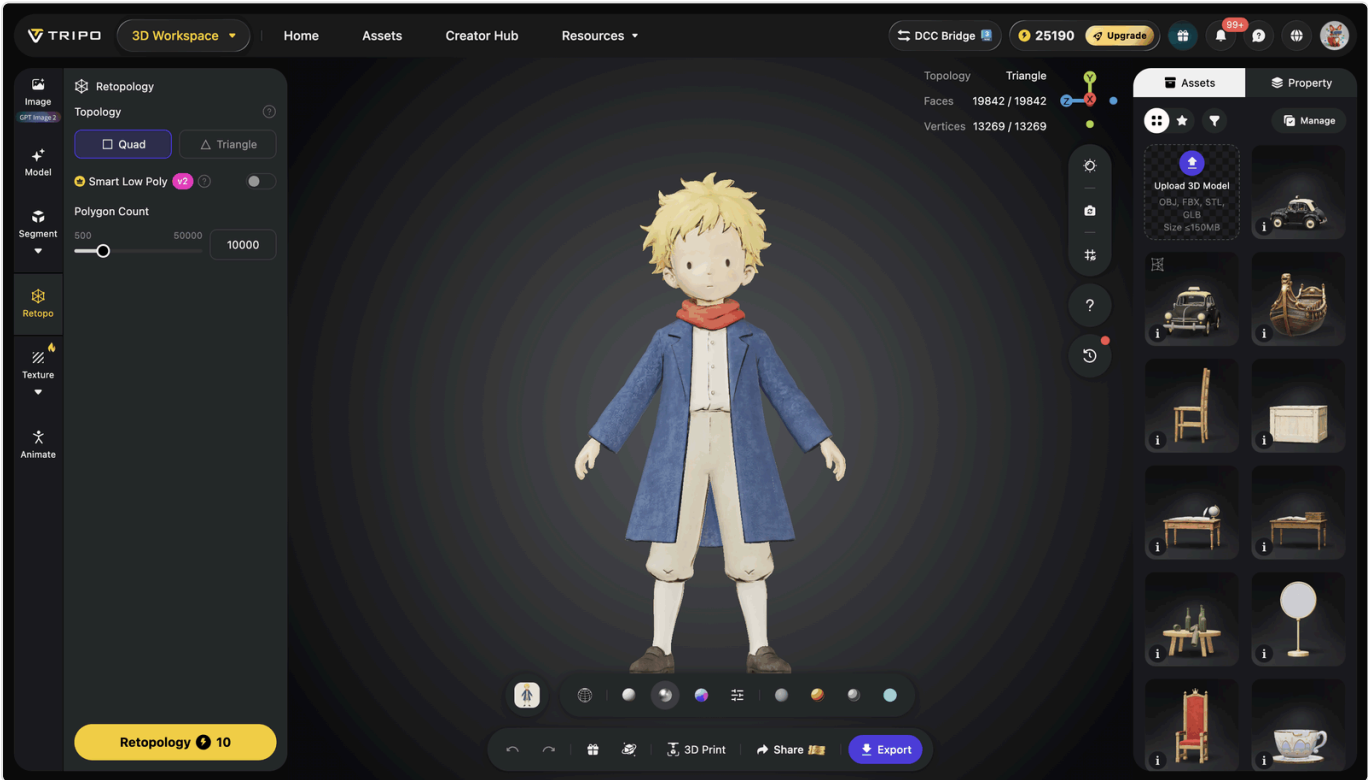
那个弹层的全貌在 § 01。这里只取两个读数：**Topology默认Triangle**，面数控制滑块量程500到2000000。还有一个数不在那张图上—— § 01那一屏的滑块被人拉到了上限，不是默认落位——这个滑块的默认值是**20000**，见 § 01那张逐项开关对照表。

默认20000这个数，正好解释了为什么我那批资产普遍落在1.5到2万面：它不是随机的，它就是表单里那个默认值。弹层顶上还有一句「We'll automatically save your settings for next time」，意思是你改一次，后面都跟着走。

智能网格那一侧是另一套默认值（弹层见 § 01）：**四边面默认选中，Polycount量程500–25000、默认5000**。两边的默认值差了四倍，这件事在你按下生成之前就已经决定了一半。想要低模，改这个滑块，别在提示词里写「low poly」——提示词管的是形状和风格，管不了这个数字。

重拓扑这一页，每个开关是什么

左侧工具栏第五个是重拓扑。选中Assets里的模型，或者直接上传一个（OBJ / FBX / STL / GLB，150 MB以内），左边面板就三样东西。



Retopology面板：Topology二选一（Quad被选中）、Smart Low Poly带v2角标的开关（默认关）、Polygon Count滑块量程500–50000、数字框默认10000，底部黄色按钮「Retopology 10」。载入的是我做的小王子角色，右上读数Triangle / Faces 19842 / Vertices 13269。2026-09-11。

控件	取值	怎么选
Topology	Quad / Triangle	要进Blender手改、要绑骨做动画，选Quad；纯粹当静态道具塞进引擎，Triangle就够

控件	取值	怎么选
Smart Low Poly	开关，带v2角标，默认关	要的是明确的低模风格时打开；我这一批没用它，所以我不下结论
Polygon Count	500-50000，默认10000	比生成表单的上限低20倍，摆明了是用来往下压面数的，不是用来重做一个模型

底下那个按钮写着「Retopology 10」，10就是积分数，跟你把Polygon Count拖到500还是50000没关系——滑块怎么拖，按钮上都是同一个数。

注意

但它跟Topology那一栏有没有关系，我没验完，如实写在这儿。我截到的两屏（这张面板图、还有后面鬼屋那次实践）都是Quad被选中，按钮上都是10，余额也确实扣了10。官方FAQ的积分表把这10拆成Retopology 5 + Quad 5，照这个拆法，Topology选Triangle时按钮上应该是5——但那一屏我没单独截过。这和拆件「官方表里写5、界面按钮写40」是同一类没对完的账（见 § 07）。以你按下之前按钮上显示的那个数为准。

10积分和50积分，什么时候按哪个

拿它去比的那一头是重新生成：再跑一次高精度模型，界面默认标50，其中35分是8K贴图，把档位拨到2K是30（这笔账在 § 07）。

判断标准其实很简单，看你不满意的是什么：

推荐

形状是对的，只是面太多、或者拓扑乱得没法绑骨 → 走重拓扑，10分，形状不会变

不推荐

形状本身就不对（比例歪了、结构缺了、理解错了物体） → 重拓扑救不了，回去改参考图重新生成，30到50分

重拓扑有它的代价，这条我实测撞过：那栋奥斯曼建筑压到2万面档位之后，精细的镂空阳台栏杆糊成了一团，细节只能靠贴图补回来。纯建模层面，2万面撑不住那种装饰性结构。面数预算的物理限制就摆在这儿：2万面撑不住那种装饰性结构，想留住它，得靠贴图而不是几何。

四边面的面数怎么换算成三角面：系数是1.75到1.8

这条坑得挺隐蔽。那栋建筑重拓扑之后，Tripo界面上读数是2.4万面，我在README里也是这么记的；后来跑E1实验解析同一个GLB文件，数出来是41106个三角面。

两个数都没错，因为界面在拓扑选四边面时数的是四边面，而导进引擎时四边面会被拆成三角形，一个变两个。做面数预算要按三角面算，别按界面读数算。

但这里有个我自己写错过、必须改回来的地方。我一度把这条换算写成「大致乘以二」，那是高估。因为所谓的四边面模型从来不是纯四边形，里面总混着一部分三角形，而三角形拆成三角形是一比一，不翻倍。真实系数

是这么来的：

$$\text{三角化后面数} \div \text{原生面数} = 1 + \text{四边形占比}$$

E10那件四边形占79.51%， $1 + 0.7951 = 1.795$ ，而实测是 $15431 \div 8596 = 1.795$ 。9月11日那辆P2.0出租车四边形占74.5%， $1 + 0.745 = 1.745$ ，实测 $9800 \div 5615 = 1.745$ 。两次都对得上小数点后三位。

样本	原生面数	四边形占比	三角化后	实测倍数
E10 · B件 (09-12)	8,596	79.51%	15,431	1.795

P2.0出租车 (09-11)	5,615	74.5%	9,800	1.745
-----------------	-------	-------	-------	-------

记这条：四边面的面数进引擎大致要乘以1.75到1.8，只有纯四边形网格才趋近2。按2去排预算，会让你的面数预算系统性高估约10%——本来能塞下的东西被你砍掉了。真要精确，就去解一次文件把四边形占比数出来，套上面那个公式。

同一张出租车设计图走智能网格出的结果见 § 01：Topology Quad / Faces 5615 / Vertices 5381，线框能看出规整的四边面走向。

能减到多少，由几何有多光滑决定

E1这个实验的方法很笨：拿11个真实资产，每个都跑一遍 `gltf-transform simplify`，比例依次取0.5、0.2、0.1、0.05、0.02，记下面数不再往下掉的那个拐点。脚本在 `三模式实验/E1-减面极限/run.py`。表里的「原始面数」是直接读GLB的索引访问器算的：`indices.count ÷ 3`，也就是文件里实打实写了多少个三角形。

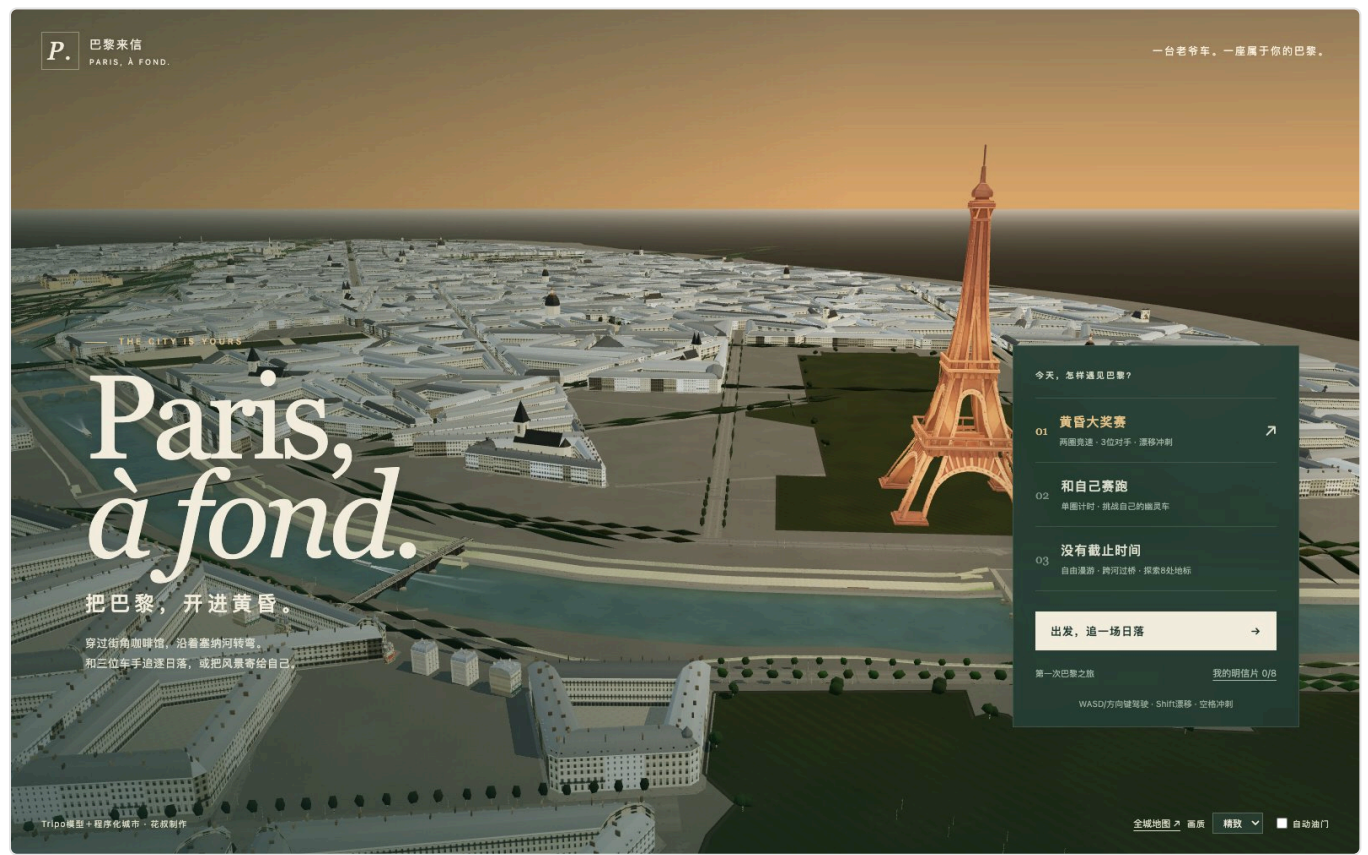
资产	类别	原始面数	减面下限	下限占比
pedestrian-01-man-suit	人物	19042	629	3.3%
pedestrian-05-painter	人物	18542	1450	7.8%
car-01-taxi	载具	18083	1633	9.0%
prop-02-bench	道具	16320	1526	9.4%
citroen-2cv	载具	18078	2001	11.1%
prop-cafe-shop	道具	17653	5648	32.0%
building-01-corner	建筑	17043	6258	36.7%
building-03-boulangerie	建筑	16696	6783	40.6%

资产	类别	原始面数	减面下限	下限占比
haussmann-building-lowpoly	建筑	41106	18270	44.4%
eiffel-tower	地标	19424	14087	72.5%
tree-01-plane-green	植被	7473	7123	95.3%

规律不在类别那一列，在几何本身：人体是连续曲面，砍到3.3%还认得出是个人；车和长椅是大块平滑面，10%上下；建筑有大量直角、窗洞、栏杆，只能到37%至44%；**铁塔那种格构结构和树那种分离薄片，几乎砍不动，70%到95%就是它们的地板。**

为什么会这样，想一下减面算法在干什么就清楚了：它找相邻的两个顶点，判断合并之后形状偏差大不大，不大就合并。连续曲面上到处都是「合并了也看不出来」的点，所以砍得动；而一根钢梁只有八个顶点，合并任何两个，那根梁就塌了。**光滑度这个词，说的其实是「有多少个可以被牺牲掉的顶点」。**

这张表里的数字是「再减就崩」的断裂点，不是拿来当生产参数用的。真跑那25个资产时我用的是人物 / 载具 / 道具0.25、建筑0.45、树和铁塔0.75，全都留了余量。还有一个口径要交代：E1的分母是已经处理过一遍的副本，不是Tripo直出的原始文件（最明显的是表里那棵梧桐，7473已经是减过一轮的，Tripo刚生成出来是18414面，见 § 17），所以这些百分比只能横向比较各类资产之间的相对关系，别拿单个数字去对你自己的模型。



「巴黎来信」demo里的埃菲尔铁塔，就是表里那个只能砍到72.5%的资产。格构结构每根钢梁都是独立的细长体，删一条就穿帮，减面算法无处下手。2026-09-14复核截图。

所以别拿一套统一参数跑全库。统一给0.25，人物那边浪费了（本可以更狠），建筑那边直接毁掉细节。

批量清洗：25个资产131秒

E2就是把上面那条结论变成一条流水线：按几何类型分配减面率，人 / 车 / 道具给0.25，建筑给0.45，树和铁塔给0.75，逐个先simplify再optimize。

```
npx --yes @gltf-transform/cli simplify in.glb tmp.glb --ratio 0.25 --error 1
npx --yes @gltf-transform/cli optimize tmp.glb out.glb --compress quantize --texture-size 512
```

指标	数值
资产数	25
体积	14600 KiB → 9433 KiB（减35%）
总耗时	131秒
平均	5.2秒/个

两条要交代清楚的：第一，那35%的体积下降里有一部分是 `--texture-size 512` 的功劳，不全是减面；第二，每个资产5.2秒里大部分是npx的启动开销而不是计算，真要批量跑，把gltf-transform装成本地依赖而不是每次npx，能快一大截。

核心建议

本书后面凡是直接写成 `gltf-transform xxx` 的命令，都是同一个工具的简写形式——装成全局或本地依赖之后就能这么敲。没装的话，在前面补上 `npx @gltf-transform/cli`，效果一样，只是每次多等几秒启动。

注意

分类不能只看文件名。 `prop-03-planter` 名字里带prop，按0.25跑却只减到7652面，因为它的花坛结构全是直角，几何上更像一栋小建筑。脚本里按前缀分类只是个近似，跑完要回头看面数对不对。

面数预算怎么定：先量，再谈数字

「多少面算多」这个问题，我原先给的是一个绝对数：一个网页demo全场景控制在30万三角面以内。回头把本书里真跑过的读数排到一起，这条经验值站不住，而且它错的方向正好是本节反复警告的那一个——**它会让你砍过头，把本来塞得下的东西砍掉。**

三组读数，全都出自同一台机器（MacBook Pro，Apple M4 Pro）：

场景	三角面	绘制调用	实测读数
小王子星球（§ 12）	147k-285k	25-56	整条渲染链GPU耗时 0.37 ms
巴黎·街上静止机位（§ 18, 09-12）	244万	77	中位渲染耗时 5.0 ms ，折算约200 fps
巴黎·C1混搭全城（§ 13, 09-10）	2,810,450	233	帧时间16.7 ms， 锁在vsync 60 fps满帧

最后那一行要读准：16.7毫秒是被垂直同步顶住的帧间隔，不是GPU真忙了多久，它只说明「没掉帧」，说不出还剩多少余量。真把余量量出来的是中间那一行——244万三角面、77个绘制调用，渲染一帧只花5.0毫秒，离16.7毫秒的预算还有三倍多。

也就是说，本书里跑得最重的那个demo，三角面是「30万」那条建议的九倍，帧率一帧没掉。所以这条经验值得换个说法：三角面不是这两个demo的瓶颈，绘制调用和贴图才是。两个佐证都在书里——巴黎那一轮把实例配额调大，三角面从144万涨到244万，绘制调用还是77一个没加，渲染耗时从2.7毫秒只涨到5.0毫秒（附录B）；而小王子那边GPU只忙0.37毫秒的时候，页面照样卡得没法看，真凶是31张贴图吃掉的5.0 GiB显存（=十进制的5.38 GB，口径见§ 12）。

所以这条建议改成：先量一次GPU耗时和绘制调用，量不出问题就别去动面数。真要一个能上手的起点，按绘制调用分档比按三角面分档有用——上面那台机器上，调用数在一百以内的场景，三角面到两百多万还剩三倍余量；调用数上到几百，先想合批和实例化，而不是先减面。这和本节结尾那条「减面之前先去量贴图」是同一个立场。

注意

这三行的日期、机位和口径互不相同，不能当成同一条曲线上的三个点。§ 18里专门交代过这件事：帧间隔和渲染耗时是两种东西，不同机位的绘制调用数天然对不上。把它们并排只为说明一件事——这三次里没有一次是三角面把机器顶住的。另外三行全部来自一台M4 Pro的桌面浏览器，换机器就得重量；移动端和低端机余量小得多，但保守的顺序仍然是先砍贴图、再合并调用，最后才轮到砍面。

面数当然还要管，只是它管的是单件而不是全场景。逐件的分配大致这样：主角和特写对象给1.5到2万（Tripo默认值正好落在这儿），中景道具5千到1万，远景和需要成百上千个实例的东西1千以内、或者干脆用程序化几何。单件实在压不下来还有个后手是LOD——同一个物体准备几套精度不同的模型，离得远时换成粗的那套。

Blender里减和命令行减，产出是一样的

E0那次实验专门比过三条路：全程Blender、直连命令行、以及两者混搭。结论对我来说挺解气：减面这件事上Blender和命令行的产出是等价的，同样的比例、同样的面数、观感上分不出来。所以如果你不会用Blender，减面这一步完全不必为了它去学。

Blender真正不可替代的地方在别处，是绑定和权重编辑那一层，那是§ 13的内容。混搭那一版的收益也很具体：多付15%的三角面，换来14栋建筑而帧率不掉。

还有一条我得说清楚：**Tripo**自带的重拓扑和本地的**glTF-transform simplify**，我没有做过同资产、同目标面数的正面对比。它们做的本来也不是一件事（一个重铺、一个删减），但「同样降到2万面，哪个观感更好」这个问题，我手上没有数据，所以不替任何一边说话。这条实验在我的待办里编号E8。

做完减面还有一件事值得先算：你打算用GLB替换掉的那些程序化物件，替换后的三角面总量是多少。巴黎那座城的行道树原先是程序化的实例，全城两万株只吃9个draw call、8.4万三角；我一度想全换成Tripo的树，近景档的实例上限是150株（这150是总配额，几个变体分摊它，不是每个变体各150），真换上15k三角的GLB，光近景一项就是225万三角，是整层预算的26倍。

程序化树和换成Tripo GLB之后的对照图在 § 17（同一组对比，那里还带着减面前后的面数）。

减面之前先去量贴图

这一节到这里全在讲怎么把面数降下来，所以必须补一句反的：你手上那个卡顿，多半不是面数造成的，减面救不了它。

前面定面数预算那一段引的0.37毫秒，出处就是这件事。做小王子星球那个demo时页面卡得没法看，我丢给agent的原话是「性能需要优化500%」，但我没让它直接去减面，先量了一次——整条后期链只吃掉16.7毫秒预算里的0.37毫秒，卡顿不可能来自三角形。真凶是31张贴图占掉的5.0 GiB显存（=十进制的5.38 GB，口径见 § 12）。只用**glTF-transform resize**把贴图降到1024²、几何一个面不动，显存就从5.0 GiB掉到0.22 GiB，三角面147,879一个没变。怎么量、命令怎么敲、逐项前后数据长什么样，在 § 12有完整一节。



降档之后的王座特写：木纹、红绒、金冠全都在，1024²在这个观看距离上肉眼无损。这张是自检截图，专门用来验证「降贴图会不会掉画质」这个疑虑。2026-09-11。

核心建议

卡顿的第一反应通常是「少画点东西」。先量一次GPU耗时，你可能会发现GPU是闲的。这次要是直接去减面、关泛光、砍阴影，画质掉一大截，而真正吃掉5.0 GiB显存的贴图一张没动，问题原封不动。所以这一节的所有减面手段，都请等你量完之后再

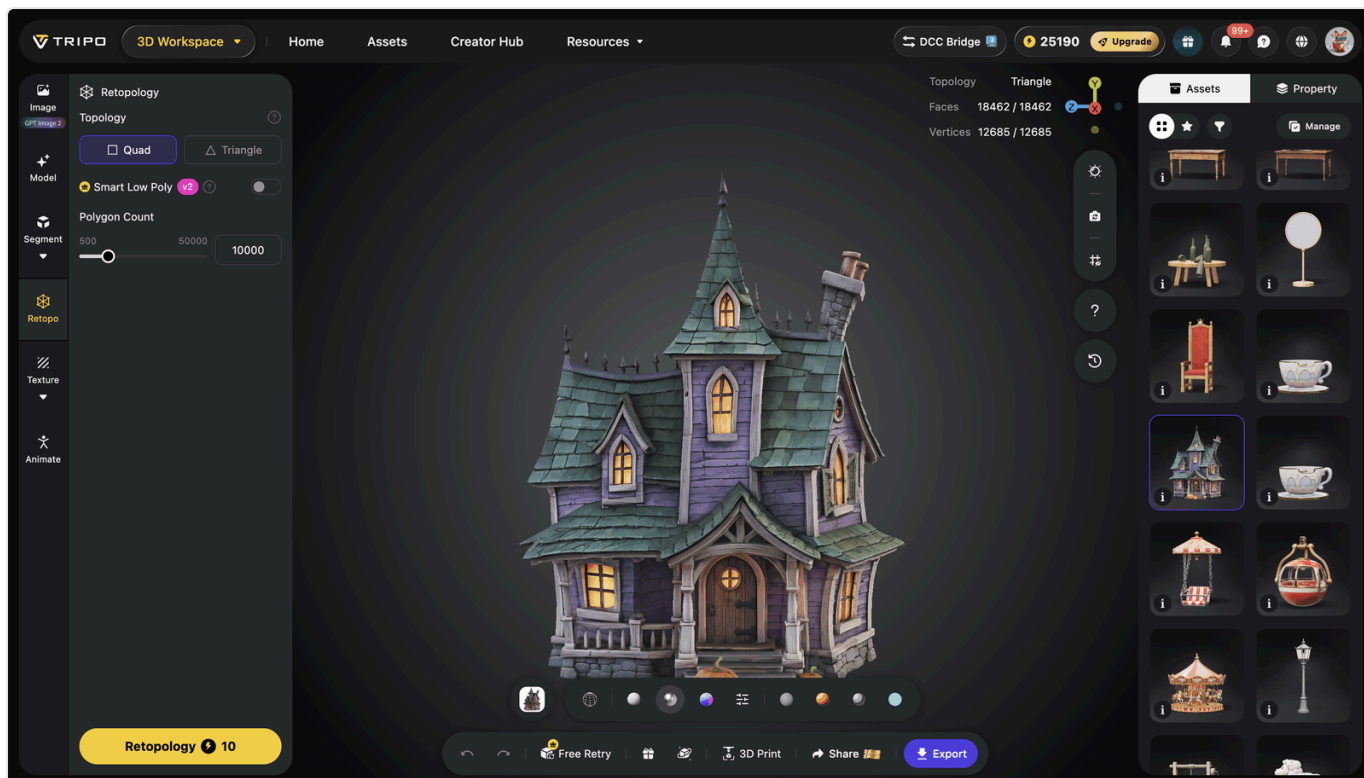
什么时候该走重拓扑

把这一节收成一张判断表：

情况	动作	成本
面数已经在1.5-3万，做静态道具	什么都不用做	0
面数几十万（镂空建筑、复杂机械）	重拓扑，Polygon Count给2万上下	10积分（我这次拓扑选的是四边面）
要绑骨、要进Blender手改	重拓扑，拓扑选四边面	10积分
一批资产整体要瘦身进网页	本地glTF-transform，按几何分配比例	0积分，5.2秒/个
加载慢、显存爆、机器风扇狂转	先量贴图，多半是resize而不是simplify	0积分
形状本身就不对	改参考图重新生成	30-50积分，看贴图档位

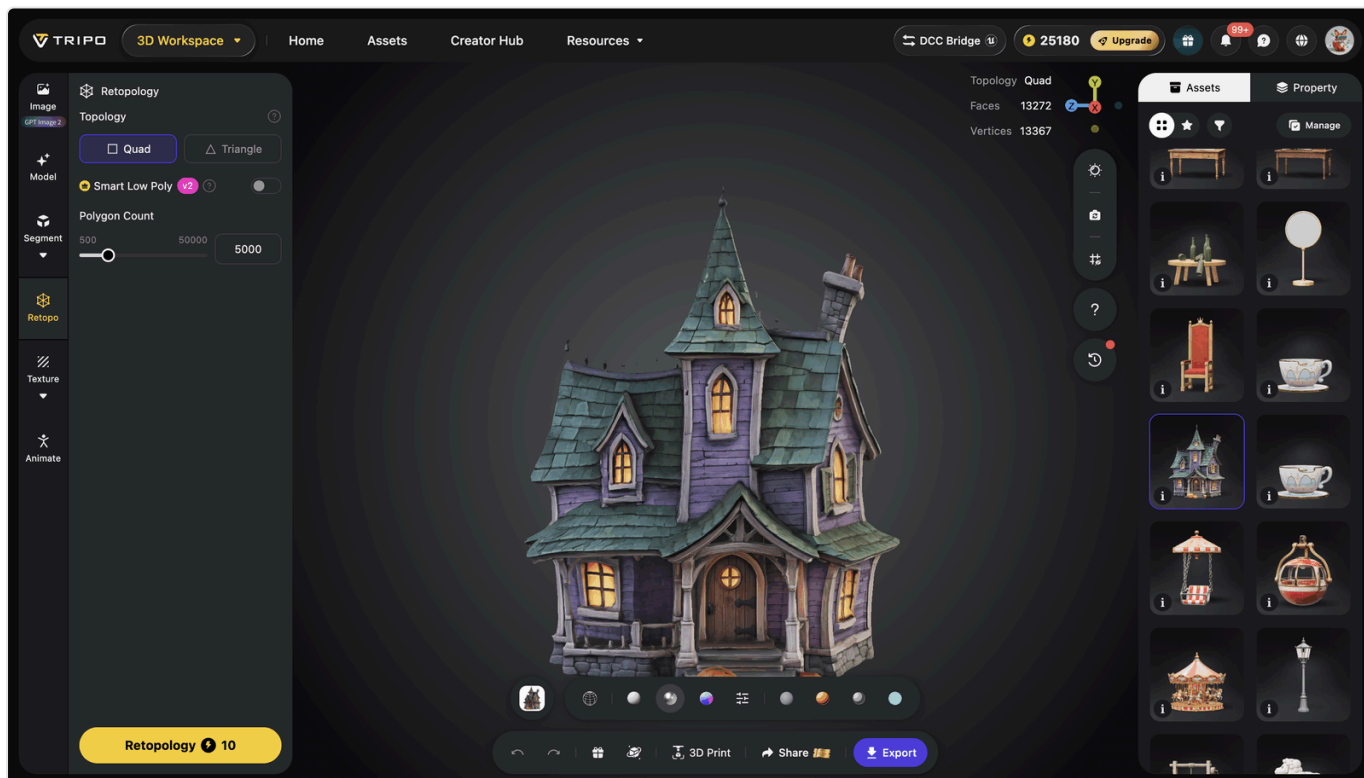
我真跑了一次，结果和我填的数对不上（已结案）

上面这些都是账面上的说法，最后我还是花10积分跑了一遍，拿一栋鬼屋当样本。它是v3.1 HD直出的，三角面18462、顶点12685。



跑之前：右上读数Topology Triangle、Faces 18462 / 18462、Vertices 12685 / 12685。左边面板Topology选Quad、Smart Low Poly v2关着、Polygon Count还是默认的10000，按钮「Retopology 10」。顶栏余额25190。2026-09-11 15:00。

我把Polygon Count的文本框改成5000，按回车，页面没有任何反应（滑块有没有跟着动我也没看清），然后按下重拓扑。等了两分钟左右。



跑完：Topology变成Quad，Faces 13272、Vertices 13367，顶栏余额从25190掉到25180，10分确实扣了。模型的外观没塌，屋顶瓦片、窗框、门廊柱子都还在。2026-09-11。

拓扑确实换成四边面了，面数却没落到我填的5000，而是13272。当时我给自己留了两种可能：要么那次输入根本没提交上去（页面确实一声不吭），要么四边面的计数口径本来就不一样。写书的时候我把它标成了悬案。

第二天这个悬案就结了，而且答案是两种可能同时成立。

第一层：输入框的状态没同步，页面一声不吭

9月12日批量跑P2.0的时候我又撞上了同一件事，这次留下了铁证。凯旋门那件，面数输入框里明明白白显示着15000，提交完去接口读回来，`face_limit` 是5000，也就是默认值。65积分就这么打了水漂。

根因是这个输入框背后是响应式框架托管的：你（或者你的脚本）把字打进DOM，`value` 确实变了，但框架内部那份状态没收到通知，提交时它拿的是自己那份，也就是默认值。界面上你看到的是15000，提交出去的是5000，中间没有任何提示。

手敲键盘一般不会触发这个，因为真实按键会派发完整的事件链；用自动化脚本往里灌值就必中。正确写法是绕过属性赋值、直接调原型上的setter，再手动把事件补齐：

```
const tb = [...document.querySelectorAll('input')]
  .filter(i => i.type !== 'file' && /\d+$/i.test(i.value)).pop();
const set = Object.getOwnPropertyDescriptor(
  window.HTMLInputElement.prototype, 'value').set;
set.call(tb, '15000');
['input', 'change', 'blur'].forEach(e =>
  tb.dispatchEvent(new Event(e, {bubbles:true})));
```

核心建议

一个不用信任何人的硬验证：设完值之后，在输入框里按一下方向键「→」（`ArrowRight`）。如果它变成15002，说明框架内部拿到的确实是15000，加二是它自己的步进。如果按完跳回5002，那你刚才那一下白设了。这一招比看界面显示什么可靠得多，因为界面显示的就是被骗的那一层。

注意

还有一条连带的：面数设置不会被「自动保存您的设置」记住。表单弹层顶上写着会帮你记住上次的配置，但面数这一项每生成一件都得重设。批量跑的时候这是最容易漏的一步，漏了就是安安静静地按默认值出一件废品。

第二层：两边的8000，数的东西不一样

第二种可能也是真的，而且就是本节开头E10那场实验验证出来的：智能网格的面数上限数四边面，高精度的数三角面。同一个数字填进去，一边给你8,596个四边面（合15,431三角），一边给你7,379个三角面。重拓扑这一页的Polygon Count是同一套逻辑——拓扑选了四边面，那个数字数的就是四边面。

套回鬼屋那件：默认值是10000（四边面口径），实际跑出来13272个四边面。按前面那条 `1 + 四边形占比` 的公式，如果它接近纯四边形，进引擎大概是两万四到两万六千个三角面，和原来那个18462三角面的版本比，面数其实是涨的。我以为自己在减面，其实是在换拓扑，顺手把三角面数推高了。

所以这件事的完整答案是：我填的5000压根没提交出去，跑的是默认的10000；而这个10000又是四边面口径，不是我心里以为的三角面口径。两个偏差叠在一起，就是那个13272。重拓扑这一步达成的是拓扑目标（换成四边面），没达成面数目标，而且如果你真正的目的是压三角面数，这一步的方向反了。

给你的操作建议就三句：按下按钮之前，先用ArrowRight验一次那个数进没进去；按下按钮之后，回头对一眼右上角那三行读数；心里记着四边面那一列的数还要乘上1.75到1.8才是引擎看到的数。这三件事加起来花不到十秒，能省掉一次65积分的空跑。

§05 智能拆分与贴图生成

Part Segmentation and Texture Generation

这一节解决两个问题：怎么让一辆车的轮子真的能转，以及怎么用同一个几何做出一整条街不重样的房子。两件事都是在模型生成之后做的，各自有一条明确的入口和一个明确的价。

轮子转不了，不是因为没做

巴黎那个demo里有可驾驶的老爷车，开起来的时候轮子是不转的。我一开始以为是自己偷懒没写那段代码，去查才知道根本写不了。

E3这个实验是专门为这件事做的。用Blender的 `mesh.separate(type='LOOSE')` 按连通分量切开模型，理论上互不相连的几何块会各自独立出来，轮子应该就掉出来了。

资产	原始面数	切出块数	最大的一块
citroen-2cv	18076	358	431面
car-01-taxi	18083	262	655面
car-02-cargo-tricycle	18490	569	320面

切出来的是几百个100到650面的碎片。**AI生成的网格不是一整块干净的曲面，它是碎的**，而且轮子和车身在几何上是糊在一起的，连通分量这把刀根本切不到它们之间的缝。这条同时解释了另一件事：为什么Blender的自动权重绑定会塌（那是 § 06的内容），根因和这里是同一个。

想明白这一点之后，很多现象就串起来了。行业里对AI生成网格有个常见评价，说它「多边形分布均匀、边环不贴合解剖结构」，听着抽象，翻译成人话就是：这些模型是从一团体素或者一个隐式曲面上抠出来的，算法只保证外表面对，不保证内部结构有意义。**它给你的是一层壳，不是一个装配体。**

所以那18076个面里，没有任何一处记录着「这里是轮胎和挡泥板的分界」。人眼能看出来，是因为人认识车；几何算法看不出来，因为几何上那里什么都没有。

核心建议

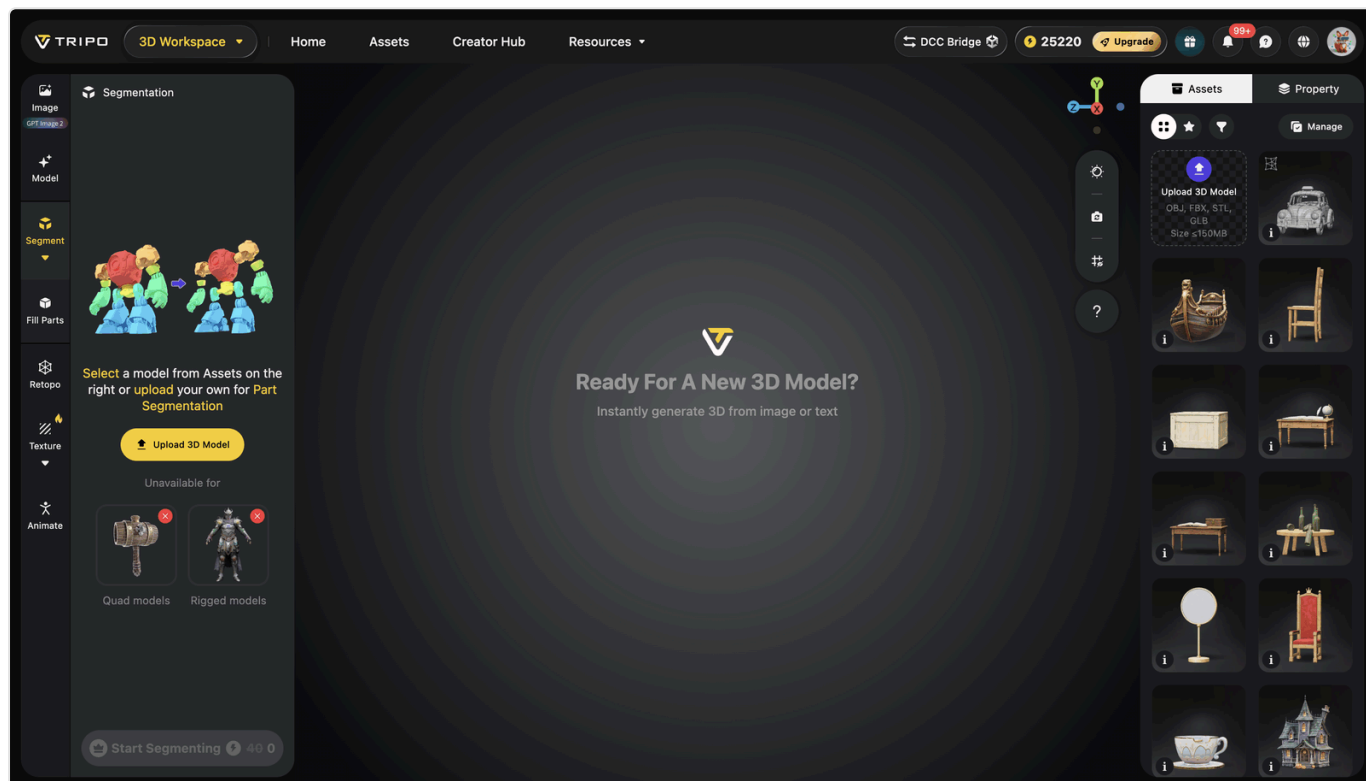
上表里2CV的18076，比 § 04那张E1表里同一个文件的18078少2。不是笔误，是两条量法。§ 04读的是GLB的索引访问器， $54234 \text{ 个索引} \div 3 = 18078$ ，也就是文件里实打实写了多少个三角形；E3读的是Blender导入之后的 `len(mesh.data.polygons) = 18076`，因为E3本来就是在Blender里切的。差在哪我回去挖了一遍：这个文件的索引缓冲里有两个三角形（从0数起排在第15733和第16071位），用的三个顶点索引和前面某个三角形完全相同，而Blender的网格不允许两个面共用同一组顶点，导入时就把重复的那两个丢掉了——顶点数两边倒是丝毫不差，都是13119。两个数都是真的：一个是「文件里写了多少个三角形」，一个是「Blender里立得起来多少个面」。差的这两个面影响不了任何结论，但既然这本书的底线是「所有面数都自己读过」，那两条量法为什么给出两个数就得写出来。§ 13那张同源的表和这里一套口径。

语义分件是什么

语义分件就是按「这块是什么」来分，而不是按「这块和那块连不连着」来分。轮子之所以是轮子，是因为它在语义上是轮子，不是因为它和车身之间有一道几何缝隙。这件事纯几何算法做不了，得有一个理解「这是一辆车」的模型来做。

Tripo Studio左侧工具栏第三个就是它，叫智能拆分，它下面还挂着一个Fill Parts。这一节只讲智能拆分，因为它是「车能不能动」这条路上唯一的关口。

智能拆分这一页，先看清两条前提



Segmentation空状态页。上传支持OBJ / FBX / STL / GLB，150 MB以内。左下角那行「Unavailable for」字号很小，但说的是最关键的两条前提：**Quad models**和**Rigged models**不在拆件的适用范围里。底部按钮「Start Segmenting 40」，这天挂着Trial所以显示划掉的40和一个0。Tripo Studio英文界面，2026-09-11。

那两条限制直接决定了你的工序顺序：

注意

拆件要的是三角面、且没绑过骨的模式。智能网格（P2.0）出的是四边面，所以要拆件，就走高精度模型的三角面版本，或者先在重拓扑里把拓扑切回三角面（按下之前看按钮上的数，这一档我没截过，说明在 § 04）。同理，一个角色既要拆件又要绑骨，拆在前，绑在后。

这条不用你去猜，Studio会当场告诉你。9月12日我录操作过程的时候，拿一个四边面模型进了智能拆分，页面直接给出提示：四边面模型不支持拆分。它在你花钱之前就把话说在前面了。所以这条规则你其实不用背，按下去之前看一眼右上角那行Topology就够了：是Quad，就知道该先绕去哪一步。

核心建议

这一节前后会出现三辆出租车，是三件不同的东西，别看混。拆件这一段的所有截图和数据，用的都是高精度那一版，属性栏读数 Triangle / Faces 18497 / Vertices 12522，它和 §04 里那辆四边面的出租车是同一张参考图生出来的两件东西。后面纹理那一段出现的则是两辆四边面的：09-11 那件 Quad / 5615 / 5381，就是 §04 里那辆；09-12 录操作过程时现场又重跑了一件，读数 Quad / 6044 / 5490。所以每张图的读数我都单独标了日期，看图先看日期。

上传之后的第一个弹层：Find the Best Angle

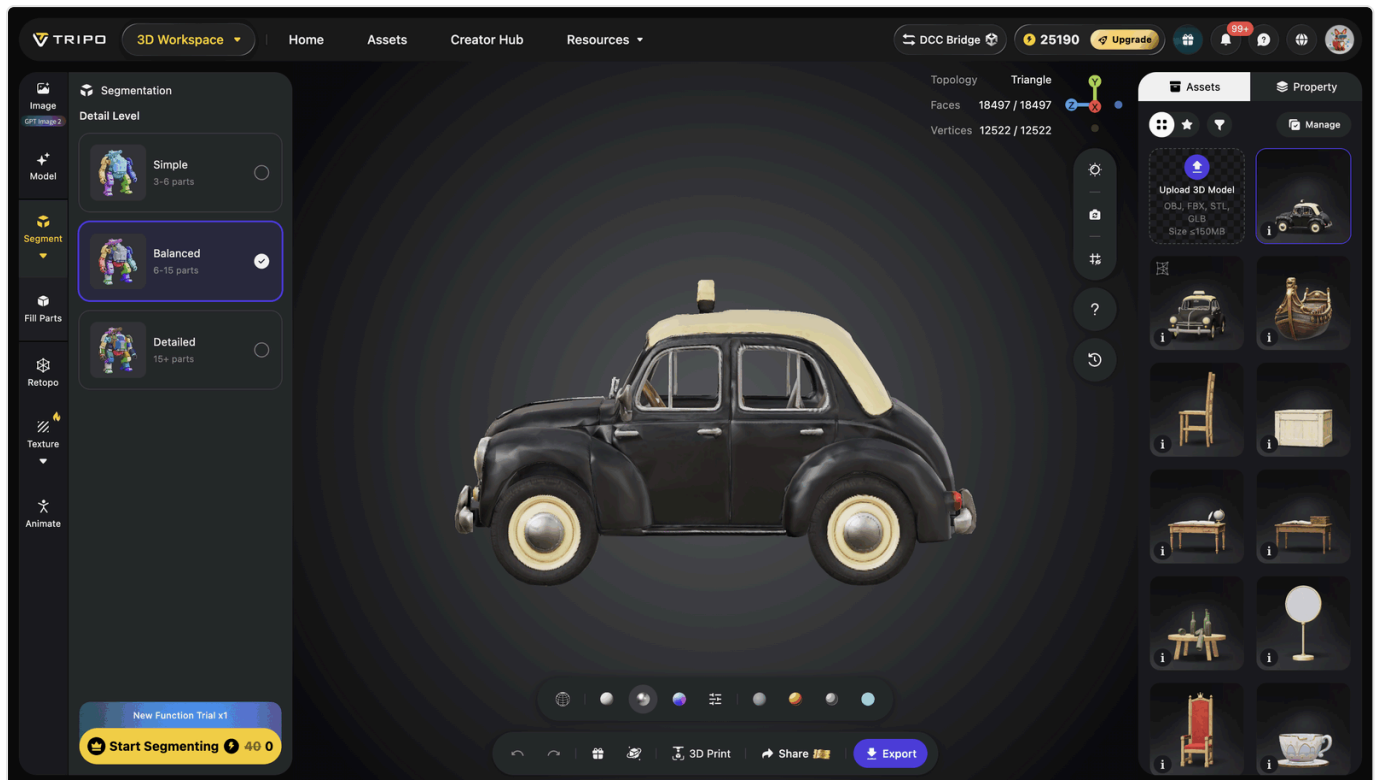
「Find the Best Angle」弹层见 §03：左栏用四组正反例告诉你什么角度算「特征充分」，右边是模型和 X/Y/Z 三个数值框。

它的提示语写的是「Adjust the view to an angle with sufficient feature information of the model」，看着像在让你调相机。但那三个 X/Y/Z 数值框动的是模型自身的旋转，不是视角。我在绑骨那一轮就是按字面理解去拖画面，结果 X 被带到了 -71.59，整个人躺平了，后面那一步直接失败。

正确做法：进来什么都别拖，确认 X/Y/Z 是 0，点 Confirm。真要转着看，用滚轮和右键，那两个操作不写进数值。「Use Original UV」保持开着，它决定了拆出来的部件要不要沿用原来的贴图坐标，开着才能让拆件后的贴图不错位。

这个弹层只在你上传外部模型时才会出现。如果要拆的东西本来就在你账号的 Assets 里，从右侧资产面板直接选中它，整个弹层根本不弹，X/Y/Z 这个坑也就不存在了。9月12日绑骨那一轮我特意走了一遍：从账号内选资产，一路顺到底，**还顺带省掉了上传外部模型要收的那50积分导入费。**所以凡是 Tripo 自己生成的东西，别下载了再传回去。

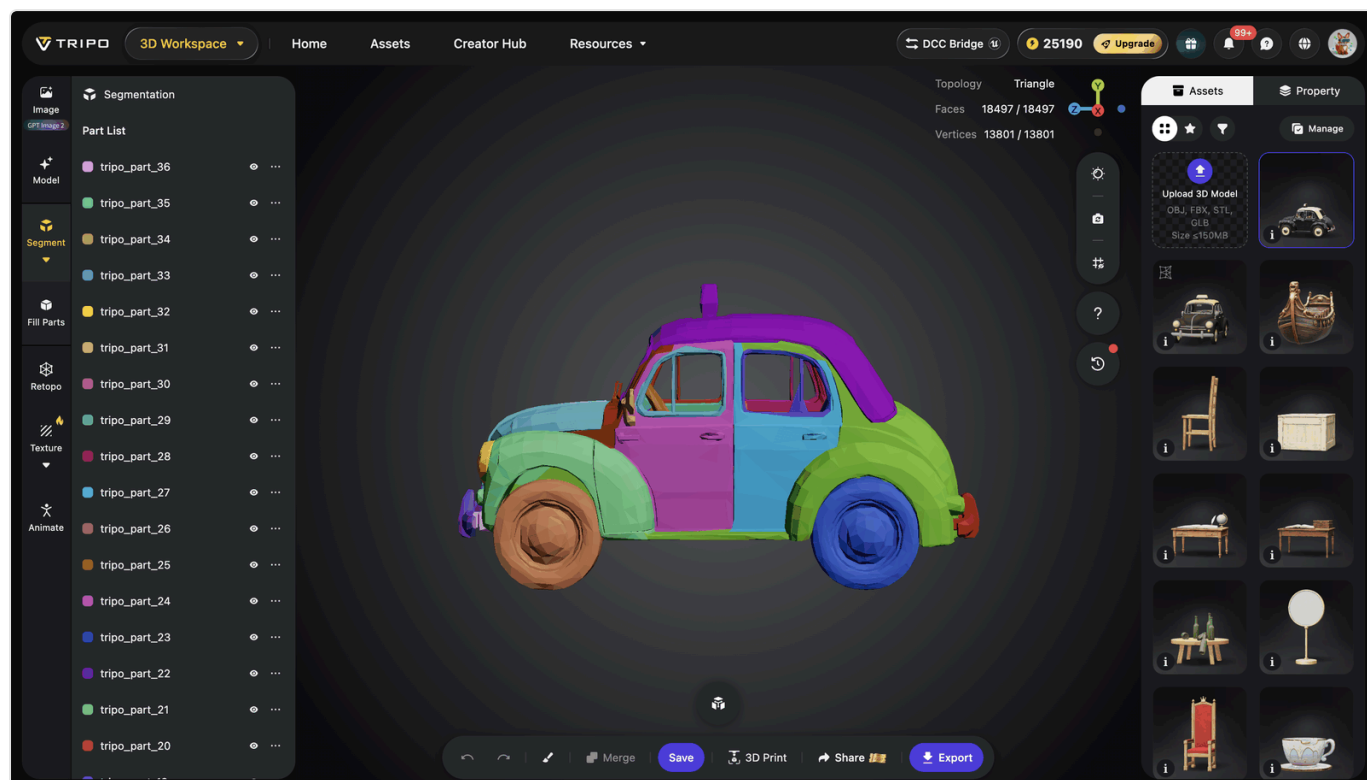
Detail Level: 三档，决定你拿到几个零件



Detail Level三档：Simple（3-6 parts）、Balanced（6-15 parts）、Detailed（15+ parts），每档配一张示意图。这次选了Balanced。右上读数是上传进来的v3.1出租车：Triangle / Faces 18497 / Vertices 12522。底部按钮带「New Function Trial x1」角标，Start Segmenting原价40、当天Trial显示0。2026-09-11。

三档给出的是部件数量区间，不是精度档。想要「能开的车门、能转的轮子」，Balanced起步；只想把一个角色分成头、身、四肢做换装，Simple就够；Detailed那一档我没用过，所以不替它说话。

结果：77个部件，轮子是独立的



Balanced档拆完的结果：左边Part List的编号从 `tripo_part_0` 一路排到 `tripo_part_36`；画面里每个部件一个颜色，前后轮各自独立（棕色和蓝色）、车门独立（蓝绿和洋红）、车顶独立（紫色）、前后保险杠也各自分开。底部工具条多出了Merge和Save两个按钮。2026-09-11。这张是智能拆分Balanced档那一次的结果，36件。后面那个77件的版本是另一条路——分部件生成的精细档，不是把拆分档位往上调出来的。

有个细节值得记：拆完之后右上角的Faces还是18497，一点没变，Vertices却从12522涨到了13801。面数不变是因为没有新增几何，顶点变多是因为切缝处的点被复制成了两份，一份归这块，一份归那块。看到顶点涨了不用慌，那正是拆开了的证据。

注意

部件数以你自己从文件里解析出来的为准，别照着界面编号写循环。这辆车的部件列表编号是 `tripo_part_0` 到 `tripo_part_76`，导出成GLB再解析，里面实打实躺着的也是77个独立网格，编号没有空缺，两个数对得上。

但这一步值得每次都核一遍：编号连续不代表网格数就等于编号数。我早先那版拆件产物就栽过——界面编号一路排到 `tripo_part_36` 看着是37个坑位，文件里却只有36个网格，`tripo_part_15` 那一格是空的；顶点数也是界面报一个数、文件报另一个数。两组数都是真的，只是量的不是同一个东西：界面报的是编辑器里的分块编号，文件里是最终写出来的网格。照着界面编号写循环，空的那一次就会取到 `undefined`。

Part List每一行右边有个眼睛图标和一个三点菜单，可以逐个隐藏、重命名、合并。Merge是把选中的几块并回一块，用来收拾那些被切得过碎的零件；Save是把拆分结果写回这个资产。

完整操作顺序

1 确认模型是三角面、没绑过骨

看工作台右上角那行Topology。是Quad就先去重拓扑切回三角面（按钮上的数见 § 04），或者用高精度模型那一版。

2 进智能拆分，从右侧Assets选中它，或上传

上传支持OBJ / FBX / STL / GLB，150 MB以内。

3 「Find the Best Angle」弹层里什么都别拖

确认X / Y / Z是0，「Use Original UV」保持开启，直接Confirm。要转着看用滚轮和右键。

4 选Detail Level，按下Start Segmenting

要可动部件选Balanced。按钮上的数字是当场生效的价，有Trial角标时会显示成0。

5 在Part List里给关键件改名，再Save

这一步做了，后面在引擎里就不用一个个试 `tripo_part_23` 到底是哪块。

拆完之后能干什么

拿到36个带名字的部件，导出的GLB里它们就是36个独立节点，引擎里按名字取就行。轮子能绕自己的轴转，车门能绕铰链开，车顶灯能单独发光。对照E3那个结论，这是同一辆车上两条路的分岔：连通分量给你358个没有意义的碎片，语义分件给你36个有意义的零件。

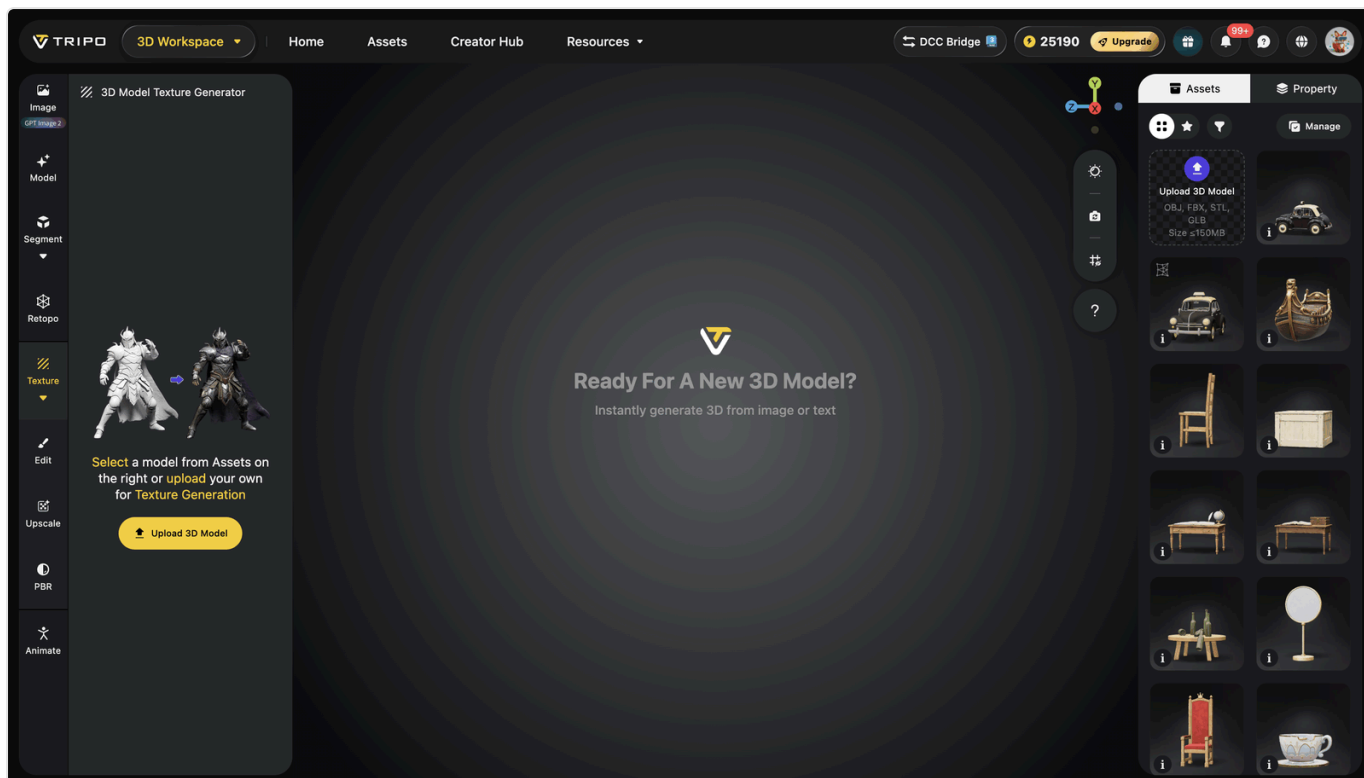
这条对教学类演示的意义比对游戏更大。一台机械要讲清楚它怎么工作，前提就是能把它拆开；我做四冲程发动机那个demo时，活塞和连杆是分两次单独生成的，因为当时还没有拆件这条路。现在回头看，**整机生成一次再语义拆分**，比逐件生成再手工装配省事得多，而且零件之间的比例天然是对的（分开生成的零件对不上尺寸，是我在那个项目里踩过的坑）。

核心建议

部件名字是 `tripo_part_NN` 这种流水号，不带语义。拆完先在Part List里逐个点一遍、把关键件改成 `wheel_front_left` 这类名字再Save，能省掉后面在引擎里一个个试的功夫。

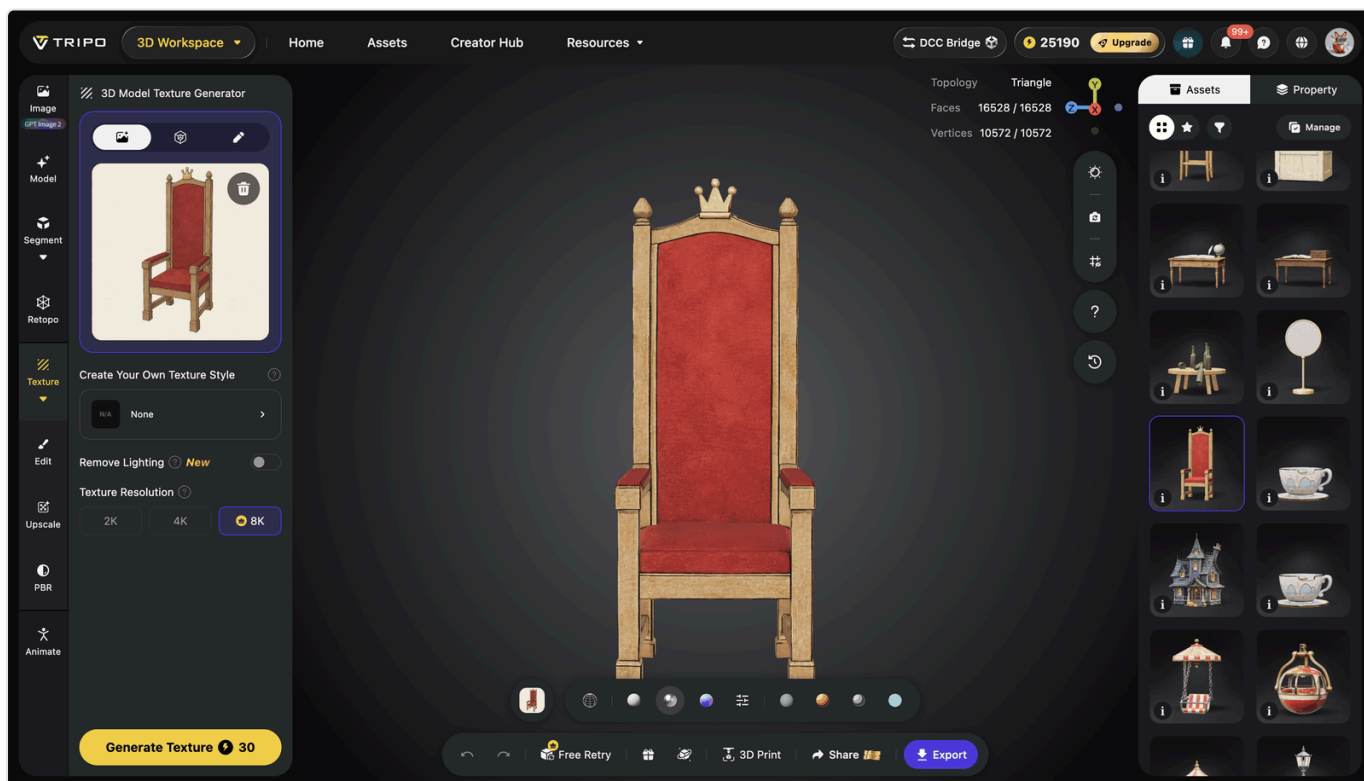
贴图生成：同一个几何，换一身衣服

左侧工具栏第六个是贴图生成。它做的事很纯粹：几何一个字不改，只重新算一遍贴图。



「3D Model Texture Generator」空状态页，示意图是一个白模骑士变成有材质的骑士。入口和Segment一样：从右边Assets选一个，或者上传自己的。Tripo Studio英文界面，2026-09-11。

选中模型之后，面板长这样：

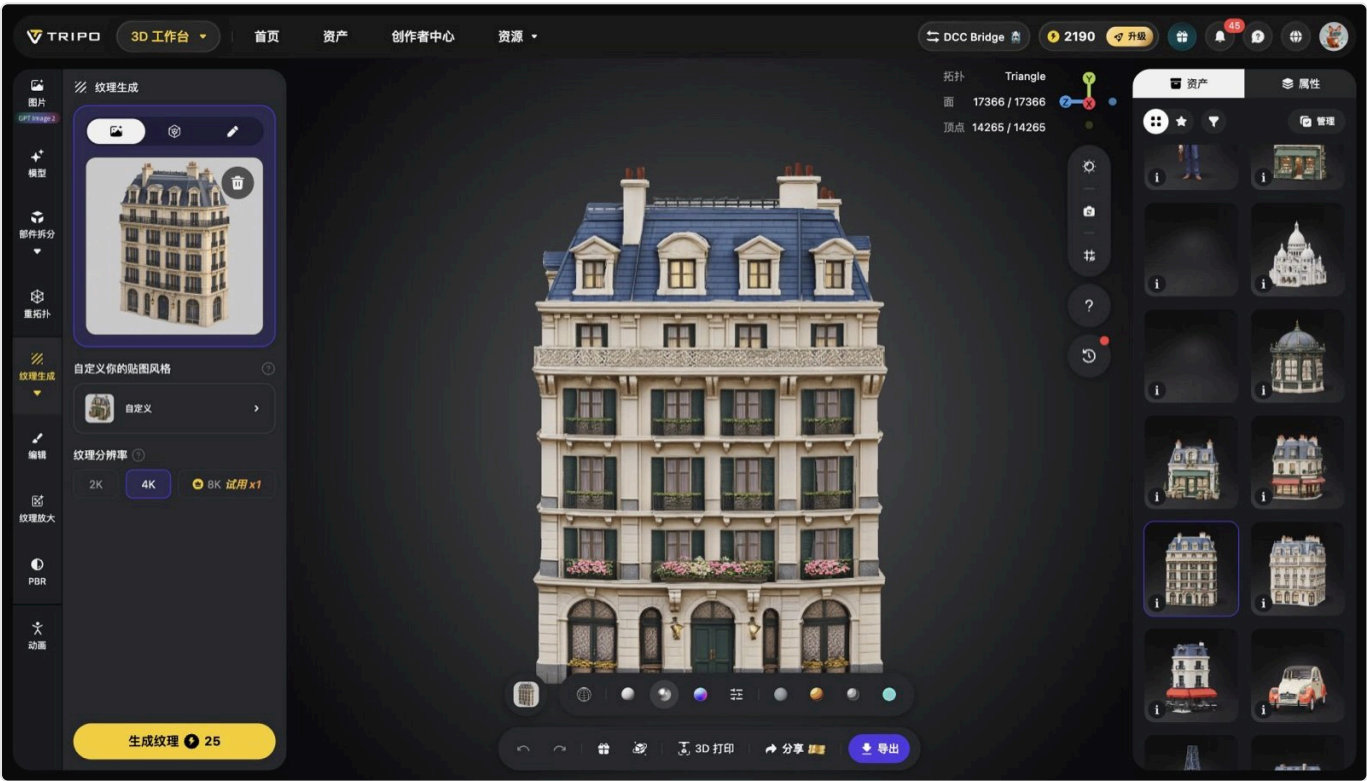


Texture面板：最上面三个页签（参考图/参考模型/画笔），下面是参考图槽位、「Create Your Own Texture Style」（这里是None）、「Remove Lighting」开关（带New，默认关）、Texture Resolution三档2K / 4K / 8K（8K带皇冠、这里选中），底部「Generate Texture 30」。载入的是我生成的王座，Triangle / 16528面 / 10572顶点。2026-09-11。

控件	作用
三个输入页签	用一张参考图定风格/用另一个已有模型的材质/直接在模型上画
Create Your Own Texture Style	把一套风格存下来复用，默认None
Remove Lighting	把参考图里烘死的光影剥掉，只留固有色。进引擎要自己打光的话该开
Texture Resolution	2K / 4K / 8K。做网页和游戏，2K基本够，\$ 04那5 GB显存就是8K来的

实测：同几何多贴图，成本是一顿早饭

这个能力我是专门去测的，因为巴黎那条街上7种建筑是循环复用的，每栋出现两次，开着车很容易认出重样。



给一栋米色石材、藏青百叶窗的巴黎建筑，用另一张店面图当风格参考重新贴图。结果：面17366对17366、顶点14265对14265，几何零改动，百叶窗变深绿、大门变绿、窗台长出花箱。左下角按钮显示这次是4K档、25积分（无参考图是20）。Tripo Studio中文界面，2026-09-10。

25积分换一栋看起来完全不同的房子，比重新生成一栋（50分）便宜一半，而且几何是同一份，引擎里还能共享几何省显存。给核心建筑各配一到两张贴图变体，一条街的重样感就没了。

这条为什么对我特别重要，得说说当时的处境。巴黎那条赛道两边的房子是七种建筑循环摆的，代码里就是 `buildingKeys[i%7]`，每栋沿路出现两次，开着车跑一圈很容易认出来。要解决它，笨办法是再生成七栋新的（350积分），聪明办法是给现有的七栋各配一张变体贴图（175积分），而且新旧两版共用同一份顶点数据，显存只多一份贴图。

API层对应的是 `POST /v3/models/texture`，官方文档写得很清楚：固定 `model_seed`、只改 `texture_seed`，就能在同一个几何上批量出贴图变体。想把这件事做成流水线而不是手点，走这条。

价格这块我得把观察如实摊开：2026-09-10那次4K加自定义参考图是25分，无参考图20分；2026-09-11再看，8K档下按钮写的是30分。按钮上的数字是跟着你当前开关实时变的，所以别背价目表，按下去之前看一眼那个数就行。

用什么当风格参考，我的经验是直接给一张同类物体的照片或设计图，效果比写文字描述稳。给建筑换贴图时我用的就是另一栋店面的图，它自带了「这个世界里的材质长什么样」这个信息，比写「深绿色百叶窗，窗台有花箱」精确得多。这和写生图提示词是同一个道理，具体方法在 § 09。

智能网格出的是灰模，贴图是必经的第二步

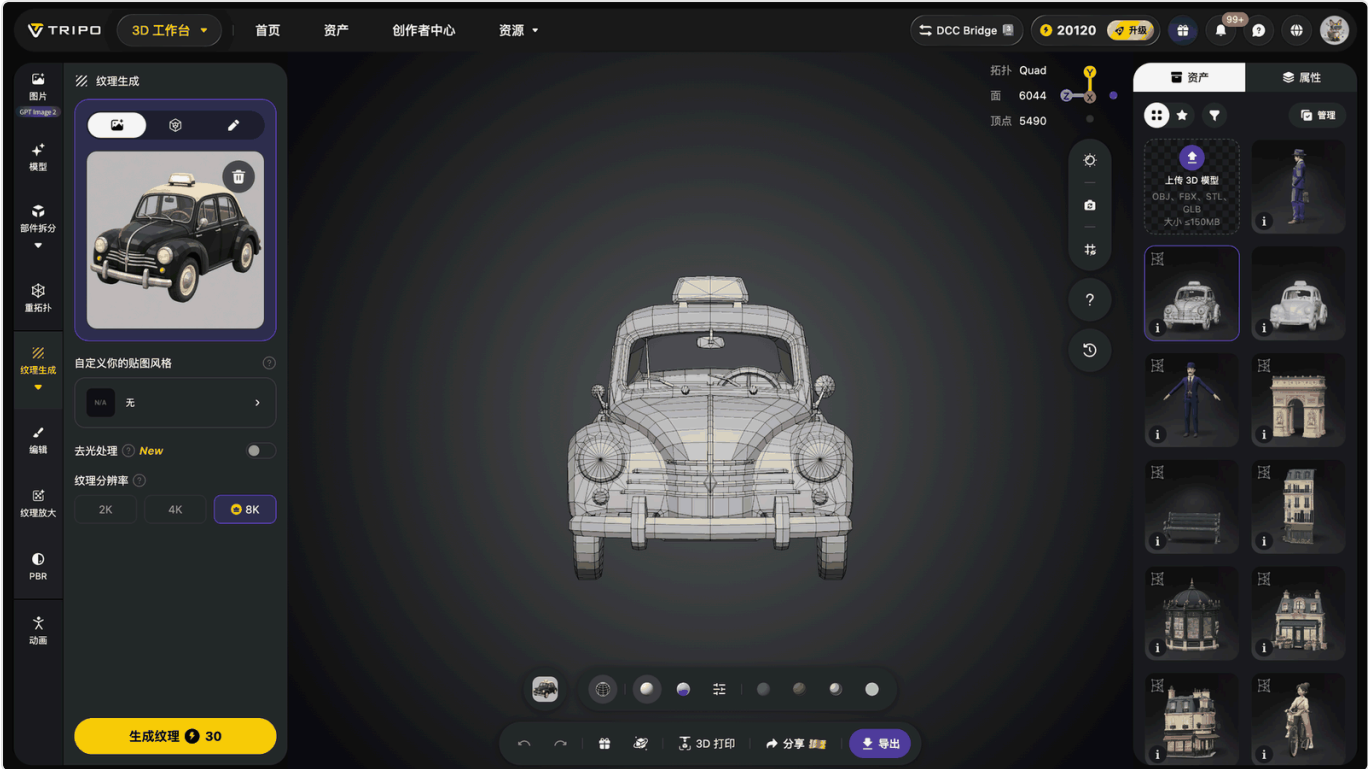
走智能网格（P2.0）这条路的话，贴图生成不是可选项。它生成完给你的是一个没有贴图的灰模，底部会弹一句「Retry for better results or proceed to Texture」。

点了Texture之后的中间状态见 § 01：中央「Generating Texture...」进度条，右侧资产卡片上跑着计时，这一次是135秒。那个数是界面自己的倒计时、含排队，不是算子耗时。

贴图完成的样子见 § 01：还是09-11那件，5615个四边面、5381个顶点，几何没动，材质上来了。这一步花了30积分，余额从25220掉到25190。

一批13件跑下来，贴图生成这一步长什么样

上面那一件是试水。9月12日我把整批巴黎资产重做了一遍，网格阶段跑了14件，其中13件接着跑了贴图生成，每件30积分，合计390积分。一次跑这么多，才看清这一步的真实形状。



纹理生成面板（中文界面）：左栏从上到下是三个输入页签、参考图槽位、「自定义你的贴图风格」（这里是「无」）、「去光处理」开关（带New角标，这里关着）、「纹理分辨率」2K/4K/8K三档——8K那一档带着皇冠，是订阅专享，并且是默认选中的。底部黄色按钮「生成纹理30」。右上属性栏读的是09-12录操作过程时现场重跑的那辆出租车（拓扑Quad /面6044 /顶点5490），和上面09-11那件5615面的不是同一次生成。顶栏余额20120。2026-09-12录操作过程时截。

这一步没有参数弹窗，点下去直接生效。不像生成表单那样会先弹一层让你确认，贴图生成是按下按钮就开跑。所以在按之前先把左栏那几项看一遍，尤其是分辨率那一档：8K是默认选中的，你不动它，出来的就是8K。要做网页或者游戏，在这里就拨到2K，能省掉后面一整轮本地压缩。

跑完之后我把参数从接口核回来，13件逐项一致：

字段	读回来的值	意思
type	texture_generation	它是一个独立算子，不是生成任务的一部分
model_version	v3.5-20260815	贴图这一步用的模型版本，和网格那一步不是同一个
texture_quality	extreme	对应界面上选中的8K那一档
texture_alignment	original_image	贴图对齐到你最初那张参考图，而不是重新想象一套
delight	true	对应「去光处理」那一项，见下面那条说明
model_url	tripo_texture_{uuid}.fbx	产物是FBX，贴图内嵌在里面

注意

deLight 这一格我没当场对上，如实记着。面板上「去光处理」那个开关默认是关的，上面那张截图里也是关的；但我这批从接口读回来的 deLight 全是 true。两边我没有在同一件东西上同时取证，所以只报读数，不下结论。真要确认，得在同一件上先记下开关状态、提交、再把接口拉回来对——这条留给复测。

逐项核一遍这件事值得做。13件里只要有一件的 texture_alignment 飘了，你拿到的就是一件风格对不上的资产，而这种错在缩略图上看不出来，要等它进了场景、摆在另外12件旁边才会现形。批量跑完不要只看「成功」两个字，把参数拉回来对一遍。

一个会让你对不上账的字段：model_version有两段

这条我卡过一次。同一个项目，网格刚跑完的时候去读详情接口，model_version 是 Nexus-v2.0-20260801；等贴图生成也跑完再读同一个接口，它变成了 v3.5-20260815。

它没有改写历史。详情接口返回的永远是这个项目当前最新的那个算子，而网格和纹理是两个算子，各有各的版本号——你读到哪一个，取决于你什么时候读。

注意

这个行为会顺手吃掉别的数据。§ 04那场E10实验里，智能网格那件的网格阶段耗时就是这么丢的：它后来跑过贴图生成，再去读接口，时间戳已经是纹理那一轮的了，网格阶段的取不回来。所以要记网格阶段的数据，就在跑纹理之前记。跑完再回头找，那一格就永远空着了。

核心建议

走P2.0这条路拿到的贴图，是一张base color，没有法线图也没有ORM。这不影响它好不好看，但影响它进引擎之后在灯光下的表现——同一盏灯下会比带三张图的版本哑一点。真正接进Three.js或者Unity的时候要要不要补，以及那张8192²的图为什么必须先降下来，在§ 12里有完整一节。

这两步要等多久

耗时这件事界面会自己告诉你，卡片上跑着秒数。我记下来的几个：智能网格那辆出租车的贴图跑了135秒，Assets页上另一个正在生成的模型显示102秒；9月12日那批里铁塔那件的纹理是109秒。

这一节的秒数都是算子耗时（算子耗时口径，不含上传与排队；端到端要往上加，两个口径的实测差见§ 01，词条见附录D）。排预算按端到端算，比较不同配置的快慢按算子耗时算，两个口径别混。

高精度模型那一侧的生成，表单上直接标着「Expect longer wait times」。这一档我没有逐件采过算子耗时，手上只有端到端的数：小王子那批16件（§ 08）和名画那两件（§ 09），都是一件约四分钟，走的都是v3.1加8K贴图。所以全书凡是写「高精度模型一件多久」，读的都是端到端那一头，和上面那几个算子秒数不是一个口径，别拿来互相换算。

这个量级意味着一件事：不要守着页面等。Pro档给的是10个并发任务，把一批要做的东西一次性都提交出去，回头再统一收，比一件一件盯着快得多。Assets页那排筛选标签这时候特别好用，Untextured那一档能一眼挑出还欠一步贴图的。

13件串着提交那一轮我还学到一条：别用「找到写着『纹理生成30』的按钮」这种方式去定位它，按钮上的文字和那个价格数字是两个独立节点，拼不到一起，等它出现会一直等到超时。按位置找那个黄色主按钮就行。同理，一次别塞太多件，我试过一口气提交三件，中途就被超时打断了，两件一批最稳。

拆不好的时候长什么样

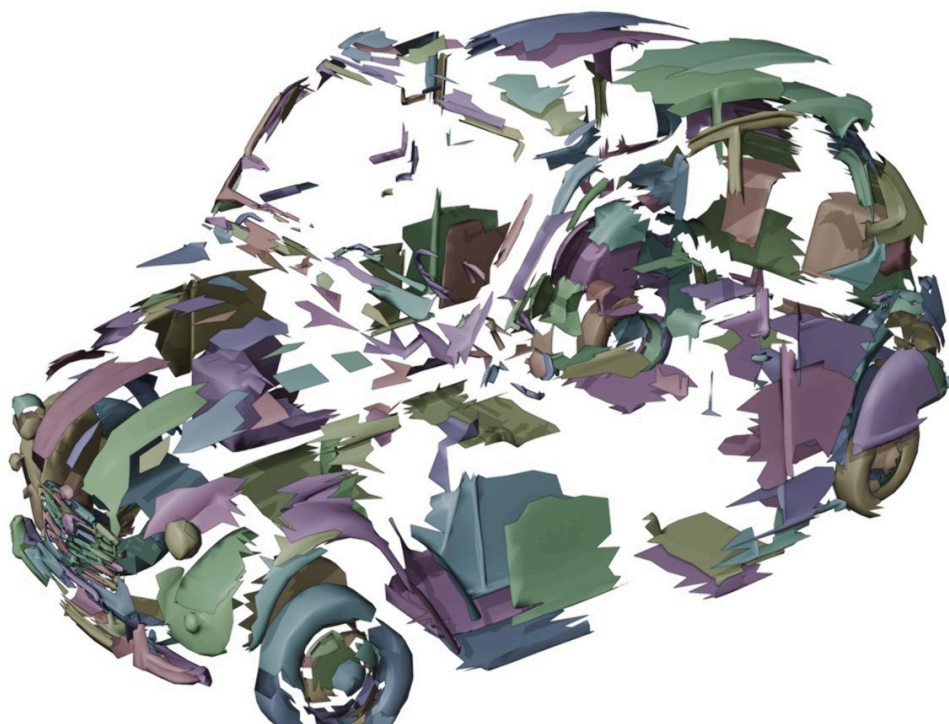
拆件不是每次都给你想要的那种分法。09-11那次Balanced档标称的是6到15件，实际出了36件，轮子、车门、车顶这些关键件都对，但也有一批很小的零碎块（保险杠上的几条、车灯周围的边）被单独分了出来。这种时候不用重跑，在Part List里把相邻的几块选中点Merge并回去就行。

重跑的话又是一次40积分，所以档位一开始就选对比事后修划算。我的经验是：想要的部件数落在6到15之间就Balanced，不要因为「多分点总没坏处」去选Detailed，碎片越多要合并的手活越多，而且从这次的结果看，Balanced本身就可能给到比标称更多的件。

两件事的工序位置

你想要	走哪条	价	前提
轮子能转、门能开	智能拆分， Balanced档	40（按钮上的数，官方表里是5，差在哪见 § 07）	模型必须是三角面、未绑骨
一个几何多种外观	贴图生成，带参考图	按档位，20-30	无
智能网格出的灰模要上色	贴图生成	30	无
角色既要拆件又要绑骨	先智能拆分再动画	40 + 20	顺序不能反
只是想省钱做一条街	一个几何+ N张贴图	50 + 25 × N	比生成N栋便宜一半

一条判断线：形变了要重新生成，形没变只是看起来不对，多半是贴图的事。这条能帮你在按下那个50积分按钮之前，先想想25分够不够。



同一辆车用「按连通分量拆」的结果：358块，每块沿「自身质心减整体质心」的方向外推0.3，随机配色。和上面Segmentation拆出的36个部件放在一起看，差别一目了然——连通分量不认识「车轮」，它只认「这堆三角形彼此连着吗」，于是一颗螺丝、一块挡板、一片窗框各算一块。Blender 4.4.3 headless跑 `separate(type='LOOSE')` 渲出，2026-09-11。

「拆完就能各转各的」这句话，我后来把它验了。把那个77部件的GLB接进一个最小查看器，让四个轮子各绕自己的轴转45度、车身不动：



四个轮子各绕自己的轴转45°，车身纹丝不动——拆件的价值就在这一下。2026-09-11实拍。



同一个模型，四个轮子沿车宽外推0.3。机位抬到27°才不会让远侧两个轮子躲在车身后方。

注意

拆出来的部件没有名字。77个部件全叫 `tripo_part_N`，一个语义标签都没有，所以「哪四个是轮子」只能从几何里读：贴着地、在四个角、薄的那一轴是车宽方向、断面近圆。按这套判据筛出19个零件——一个轮子不是一块，而是轮胎、轮辋、毂盖4到5件套在一起，得先按象限把它们聚成四簇，再整簇去转，否则一转轮辐就散架。四簇分别是：左后 1/35/41/57/73，右后 8/36/44/63/68，左前 25/37/39/66，右前 2/15/40/53/74。你要做的如果是「点一下车门就打开」，这一步的活儿躲不掉——拆件给你的是可动的零件，不是认识自己的零件。

§06 绑骨与动画

Auto-Rigging and Animation

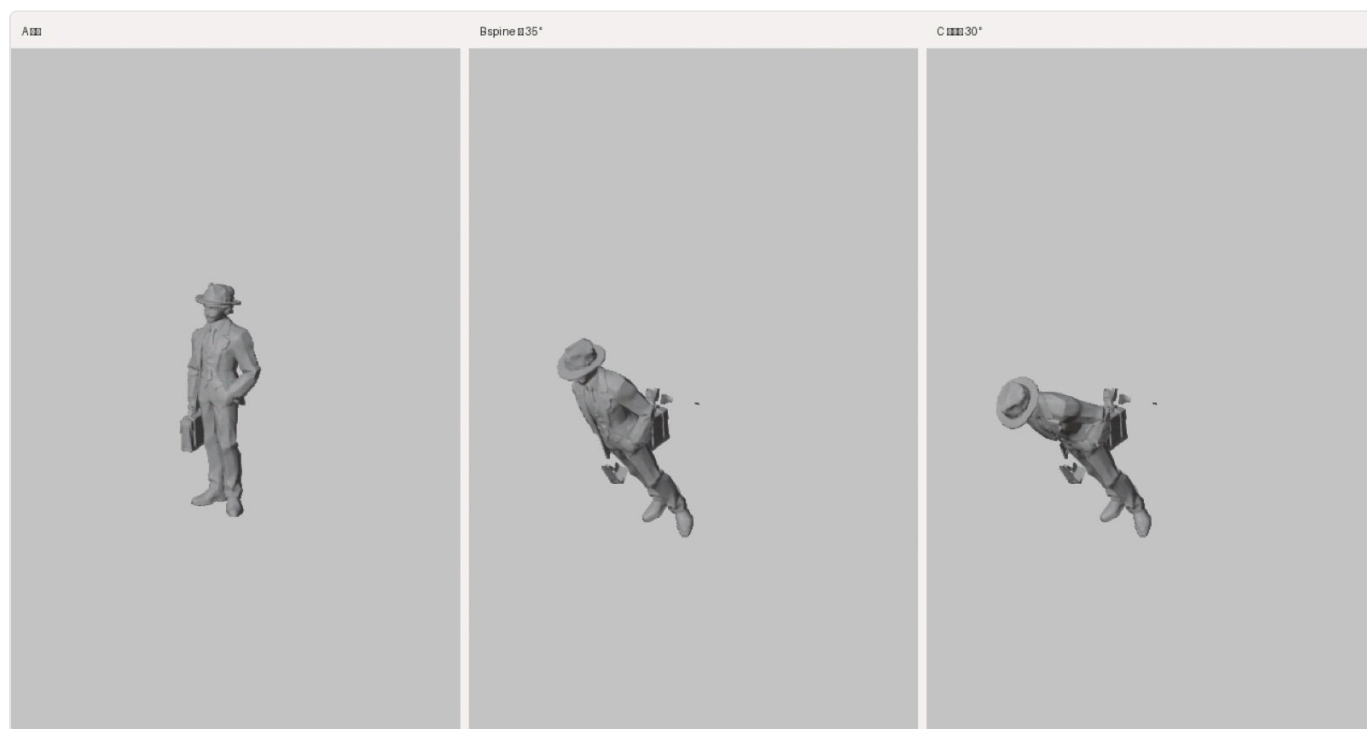
这一节走完，你能拿到一个真会走路的角色：41根关节的骨架、一段2.375秒的行走循环，导出成GLB直接进引擎。中间几个坑我都踩过，位置和判据都写在下面，包括一条能省掉50积分的入口选择。

先看反面：自己动手绑，会绑成什么样

第一版巴黎demo跑起来，街上的人走路是滑的，像脚下踩着冰。那18个行人确实是静态网格在平移，加了点上下起伏假装走路。

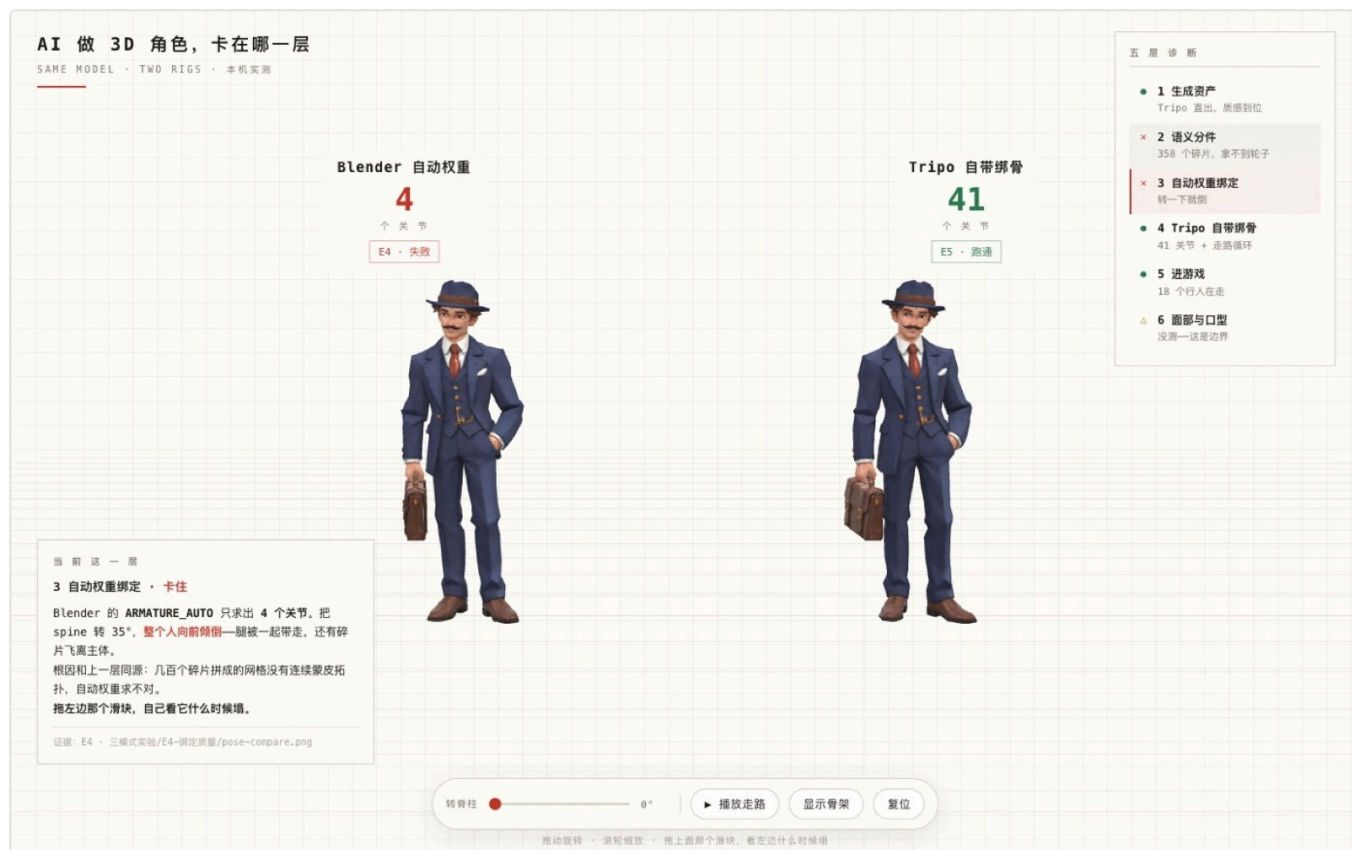
我第一反应是在Blender里给它绑。Blender有个 `ARMATURE_AUTO`，塞一副骨架进去，让它自动算权重。蒙皮权重就是「哪根骨头动的时候，这块皮跟着动多少」的那张表，绑骨的一大半功夫都在这上面。

自动权重确实跑完了，也确实生效了，结果是这样：



E4实验的三张workbench渲染：A静止是正常站姿（戴帽、西装、手提公文包）；B把 `spine` 转35°，整个人向前倾倒，腿被脊柱的权重一起带走；C再转 `head` 30°，整个人翻倒，右边还有一小块碎片飞离主体。脚本 `三模式实验/E4-绑定质量/pose_render.py`，2026-09-10。

关节数是关键线索：**Blender自动权重只求出了4根关节**（`root / spine / head / neutral_bone`），而这个网格是几百个碎片拼出来的，没有连续的蒙皮拓扑，权重自然求不对，碎片还会在变形时脱离主体。根因和 § 05那358个碎片是同一个。



把两种绑定放进同一个页面，共用一个「转脊柱」滑块：左边Blender自动权重4个关节，右边Tripo自带绑骨41个关节。右上角是五层诊断，第6层「面部与口型」标的是没测，所以不下结论。2026-09-11复核截图。

几个词

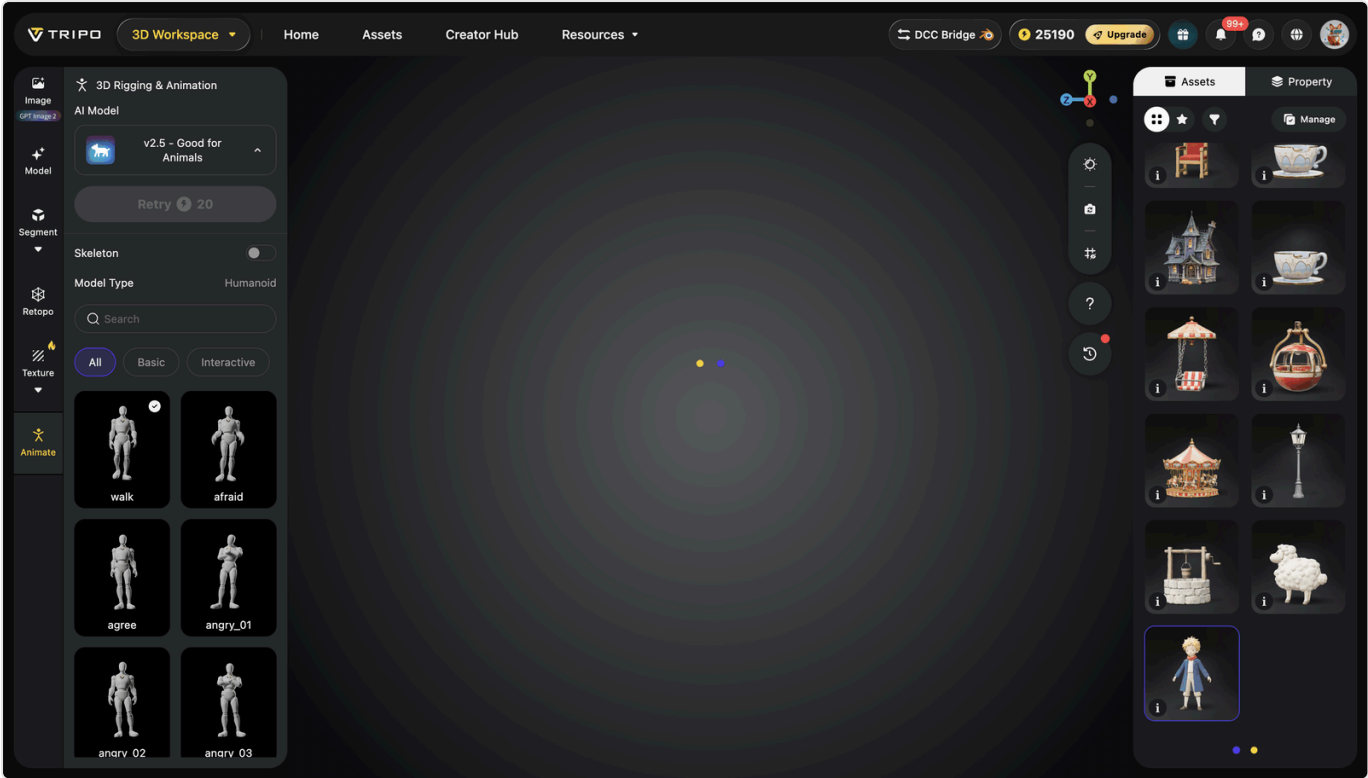
绑骨 (rigging) 是给一个静态网格装一副骨架，并算清楚每块皮归哪根骨头管。**关节**就是骨架上的节点，人形骨架一般四十来根。**T-Pose**是双臂平举成T字的中立站姿，业内把它当默认起始姿势。**A-Pose**是手臂自然下斜三四十度的站姿，看起来更放松。**重定向** (retarget) 是把一套动作从一副骨架搬到另一副骨架上。

T-Pose和A-Pose的区别在后面会变成一条要紧的判据，先记住它们不是一回事。

我原本以为这条链很长：生成→绑骨→找动作库→重定向→导出。实测下来它短得多，因为Tripo把动作库那一段也包了，不需要重定向。

入口，和那个必须改的下拉

左侧工具栏最后一个动画，页面标题叫「3D Rigging & Animation」。



Animate页面板：AI Model下拉默认是「v2.5 - Good for Animals」，下面是Retry 20、Skeleton开关、Model Type（这里识别成Humanoid）、搜索框、分类标签（All / Basic / Interactive / …），再往下就是动画卡片网格，第一个是walk。Tripo Studio英文界面，2026-09-11。

注意

做人物，第一件事是把这个下拉从「适用于动物」改成类人预设。默认值是动物骨架，直接拿去绑人会失败。更麻烦的是：刷新页面之后它会重置回动物，所以第二次重绑要重新选一次。上面那张图里下拉还写着动物，就是绑完刷新过页面的状态。

先选对入口，这一步就能省50积分

绑骨页的模型从哪来，有两条路，价钱不一样。

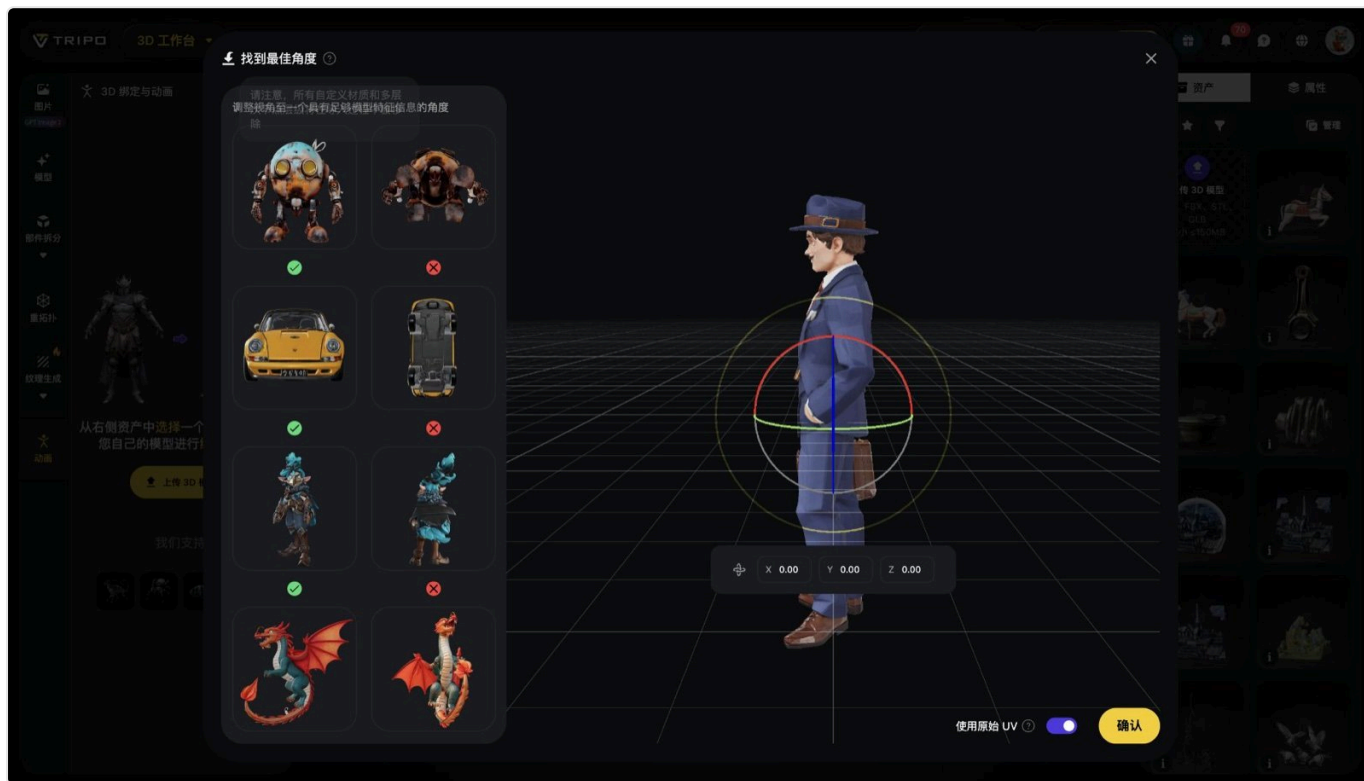
入口	额外花费	会不会弹「找到最佳角度」
从右侧资产栏里点一个自己账号内已有的模型	0	不弹
点「上传3D模型」传一个外部文件进来	50积分导入费	会弹

2026-09-12那次我绑的行人是前一天在Tripo里生成的，本来就躺在账号里，所以在 /workspace/rigging 页右侧直接点中它就开工了，既没付那50分，也没遇上下面要讲的那个弹层。这条路我后知后觉，早两天知道能少花一笔。

核心建议

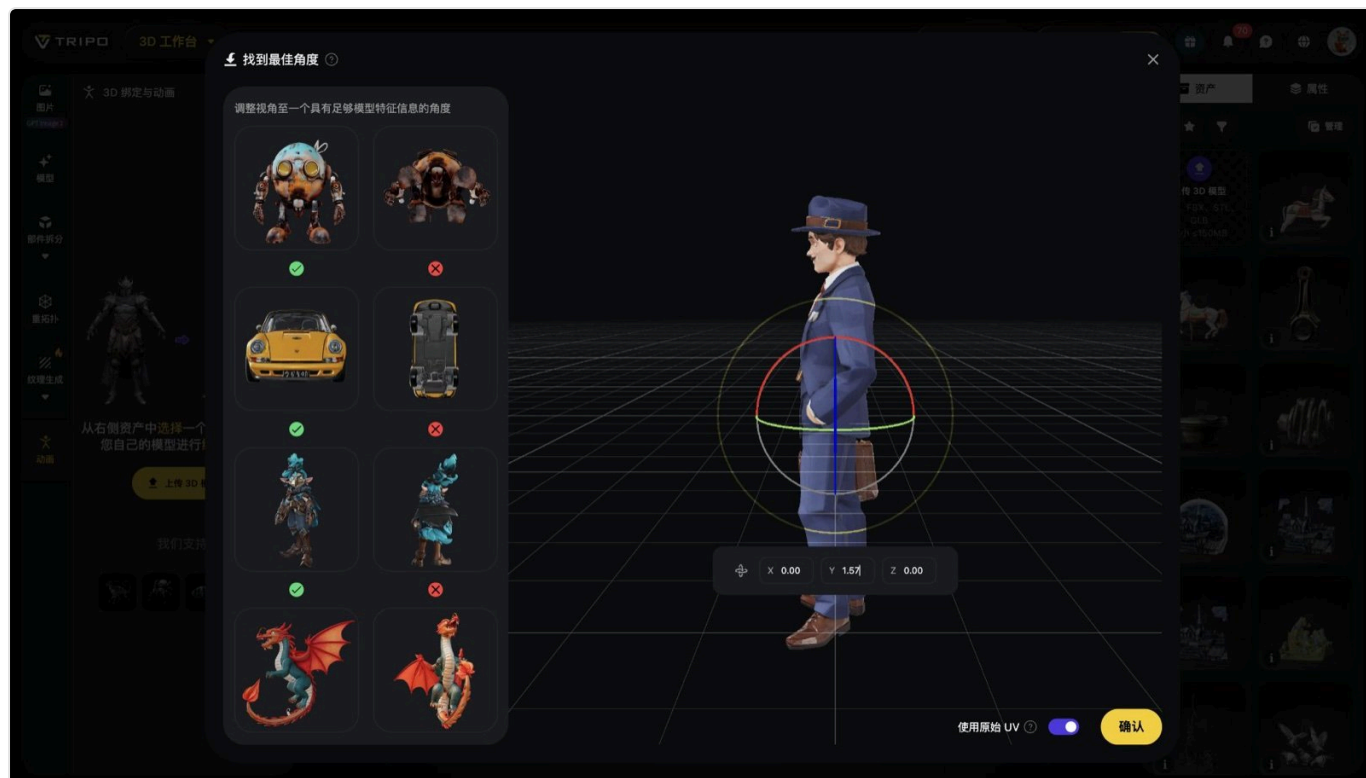
判断很简单：**模型是Tripo自己生成的，就别导出再传回去。**只有当模型是Blender、别的工具或者别人给你的文件时，才需要走上传这条路，那50分买的是把它导进Tripo资产库这件事。

如果你确实要上传：那个会把模型转歪的弹层



只有走上传这条路才会见到的「找到最佳角度」弹层，X/Y/Z三个数值框全是0.00，这是正确状态。右下角「使用原始UV」保持开启。这次上传的是巴黎那批走路姿态的行人，2026-09-10。

弹层上那句提示是「调整视角至一个具有足够模型特征信息的角度」，看字面像是在说相机。但那三个数动的是**模型自身的旋转**。我按字面理解去拖画面，数值就被带跑了：



拖了两下画面之后，Y已经变成1.57，模型被整个转了90°。我另一次更惨，X被带到-71.59，整个人前倾躺平，绑骨直接失败。正确做法：进来什么都别拖，确认三个数是0，点确认。要转着看用滚轮和右键，那两个不写数值。2026-09-10。

真正的入口条件：四肢和躯干得分得开

我头两次绑骨都失败了。第一次用的是巴黎那批「走路中被拍到」的行人（前倾、侧身），第二次换了个站直的碎花裙角色，站直了但一只手搭在胸前，还是失败。

第三次跑完，产品自己弹了一条提示出来：



页面顶部横幅：「为了更好的绑定效果，请上传T-Pose图像并重新生成模型。」右边资产栏里那个躺平的小人，就是被X=-71.59转歪之后的那一版。Tripo Studio中文界面，2026-09-10。

看到这条横幅的当天，我给自己记的结论是「前两次不是姿态不够正，是根本不是T-Pose」。两天后这个结论被自己的实测推翻了。

2026-09-12我又做了一个要绑骨的行人，参考图生成的时候模型没听话，给的是A-Pose：手臂朝下斜了三四十度，根本不是横平的T字。我本来打算重生一张严格T-Pose的，请求超时了，就先拿这张凑合着往下走。结果是点一次「自动绑定」，19秒，第一次就成了。



这两张图生成的模型都绑成了。左边是09-10那张标准T-Pose（白T恤灰裤白鞋，双臂完全平举），prompt里的风格锚定词用的是Mixamo，因为它就是「干净可绑骨T-Pose角色」这件事的母语。右边是09-12那张A-Pose（深蓝西装三件套，手臂下斜30到40度），本来只是重生严格T-Pose的请求超时了拿来凑合的。两张的共同点不是姿势，是手臂和躯干之间那道清晰的空隙，外加全身入画、正视角、两腿分开站稳。右边那张走的是Lovart会员额度，没花Tripo积分。

核心建议

所以承重的判据是这四条，不是「必须摆成T字」：全身入画、正视角、手臂向外张开与躯干有空隙、两腿分开站稳。核心那条是空隙。四肢糊在躯干上，算法分不出哪是胳膊哪是身子，骨架就绑不出来；只要分得开，A-Pose一样能过。

官方那条横幅推荐T-Pose，这个推荐没错，T-Pose是空隙最大的姿势，成功率最稳。但它是「更好」不是「必须」，我把它读成了硬门槛，多花了一轮时间去重生参考图。你手上那张图如果已经是A-Pose，先拿去绑一次试试，比重画一张快。

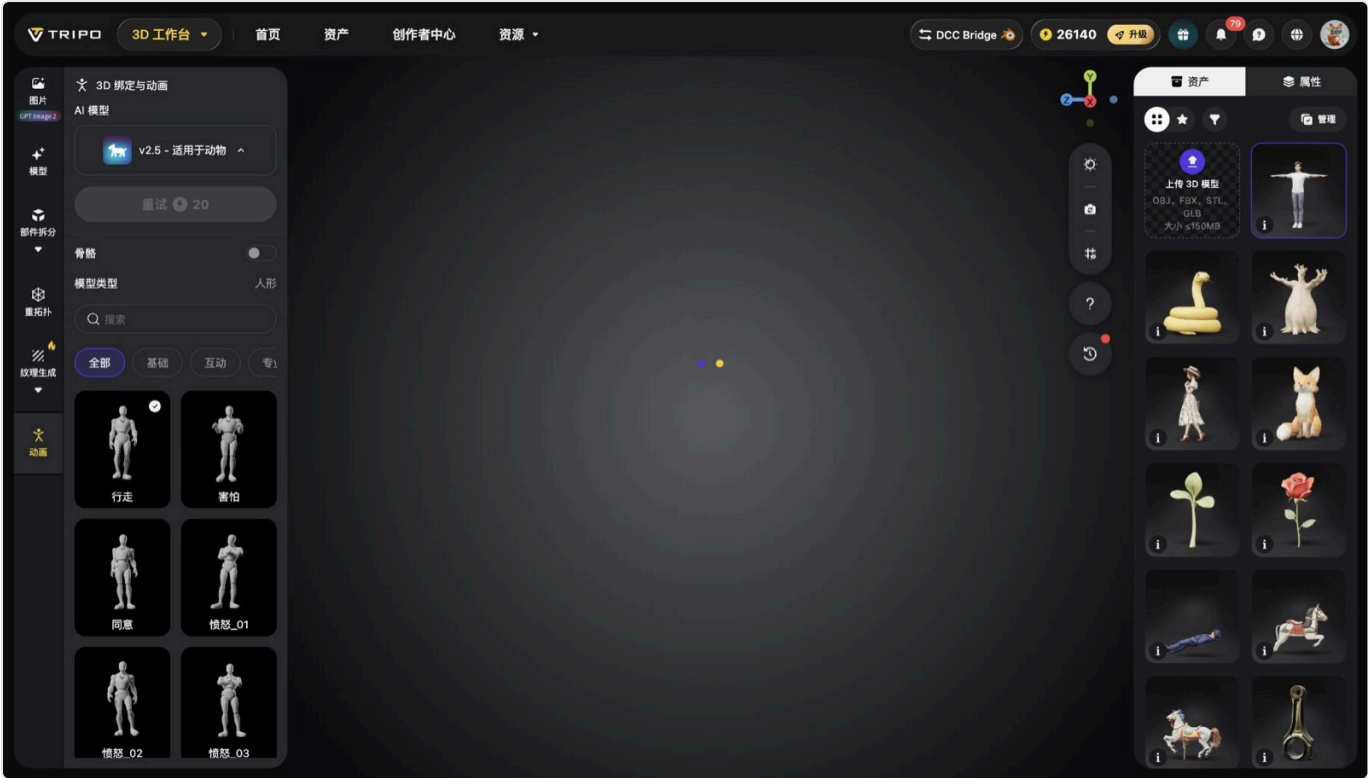
自动绑定20积分：成没成，看动画卡片上的字

选好模型、选好类人预设，点那个「自动绑定20」。2026-09-10绑那个巴黎行人时，等了一轮之后中央预览是空白的，旁边出现「重试」按钮，第二次重选类人预设再绑才成，那一次实打实多花了20分。当时我写下的结论是「第一次多半会失败，这是产品的固定行为」，这话说满了。

回头把记录对齐，样本其实只有两个，而且指向相反的方向：那个行人是第二次才成；09-12那个A-Pose行人一次就过。两个样本撑不起「固定行为」这四个字。同一天我还误判过一回，那次其实是把「确认导入」当成了绑骨，白等了一轮，并没有真的绑失败。

所以现在的说法是：第一次绑不成是有可能的，遇到了就点「重试」，不必当成坏消息；但它不是必经的一步，预算上留一次重试的余量（多20分）就够了。

判据很清楚：看动画卡片上的字。全是「请先绑定骨骼」就是没绑成；绑成功后它们会变成可点的动画名。



绑成功之后的动画库：行走、害怕、同意、愤怒_01、愤怒_02、愤怒_03……全部变成可点的名字，第一个「行走」右上角已经打了勾。左边AI模型下拉又显示成「v2.5适用于动物」，那是刷新之后的重置，不影响已经绑好的骨架。2026-09-10。

绑一次骨要多久：三个数，三种口径

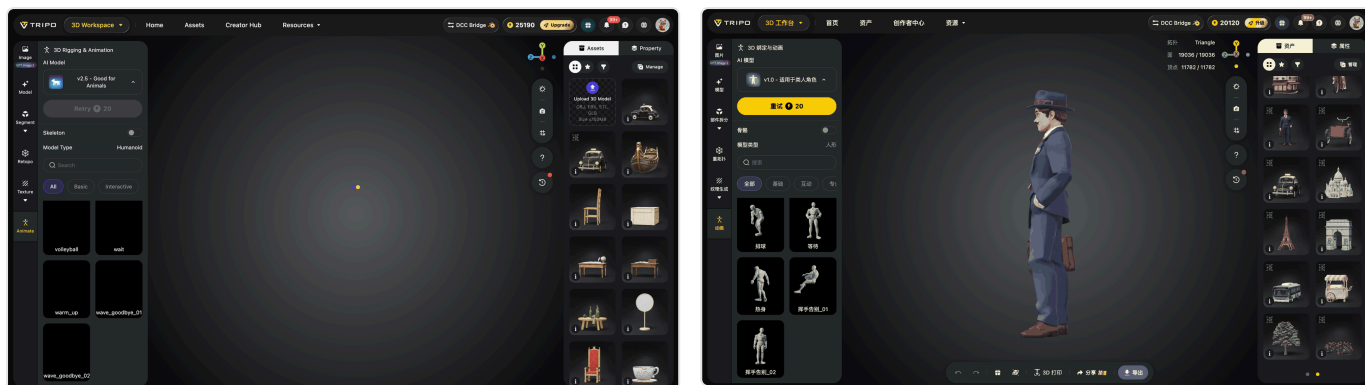
这件事我得说细一点，因为你拿秒表掐，掐到的数和界面告诉你的数对不上，不是你掐错了。

数	这是什么口径	怎么来的
3秒	接口报的算子耗时	2026-09-12 16:43那一单，detail接口里rigging算子的 updated_at 减 created_at
13秒	现场画面从点击到出结果	同一天的录屏，从按下「自动绑定」到「重试」按钮出现，数画面时间
19秒	另一件模型的现场耗时	那个A-Pose行人，一次过

界面和接口给你的都是算子本身跑了多久（算子耗时口径，不含上传与排队；端到端要往上加，两个口径的实测差见 § 01，词条见附录D）。写进项目排期的要用端到端那个数，别用接口报的。

动画库有多少个，我数了两遍

2026-09-11第一次数完，我记下的是110个，但没记数法，事后自己都复原不了是怎么点出来的。2026-09-12录屏那天重数了一遍，这次把过程留下了：左侧动画面板从头滚到尾，分8屏截完，逐张卡片数过去，类人骨架下是97个。所以这本书里凡是提到这个量，写的都是「近一百个」。



两次独立记录的列表尾，左边是2026-09-11的英文界面（余额25190），右边是2026-09-12的中文界面（余额20120）。两次都停在 **wave_goodbye_02** / 「挥手告别_02」，而且最后一行只有一张卡片，说明总数是单数，和数出来的97对得上。这种「换个日子、换个语言再数一遍」的核法，比记住一个数字可靠。

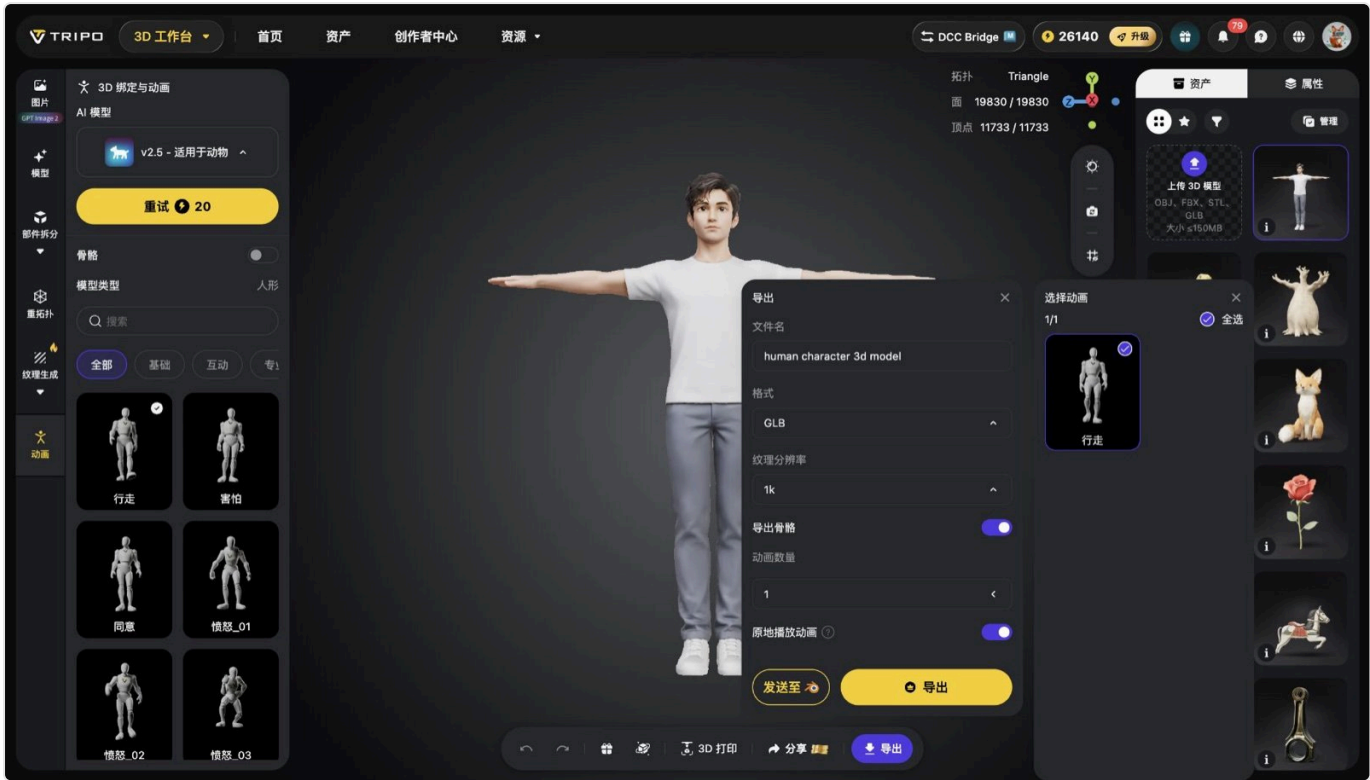
分五类：All / Basic / Interactive / Scene Specific / Emotional，另有搜索框。里面有walk、run、idle、sit、jump这些基础动作，也有dance_01到dance_06、sing_01到sing_04、greet_01到greet_04这类成组的，还有basketball_shot、surf、jump_rope这种场景动作。完整清单在 [research/studio-facts](#) 里。

这近一百个里，做游戏真正常用的其实就那么几个：idle（站着等）、walk、run、turn、sit、wave_goodbye。情绪那一组（cry、laugh、angry、frightened）在做剧情演示时有用，运动那一组更像是给短视频素材准备的。

它把动作库包进来这件事，省掉的工序比看上去多。正常流程里，你得先找一个动作库（Mixamo之类），下载一个FBX，再做重定向，把那副骨架上的动作映射到你这副骨架上，关节名字对不上还得手工连线。**Tripo**这边是它绑的骨、它出的动作，两边天生同名同构，重定向那一步整个不存在。我一开始判断「链路长、要重定向」，是错的。

最致命的一步：点动画只是预览

我第一次导出拿到的GLB是「有骨骼、零动画」，查了半天才明白问题出在哪：在动画库里点一下卡片，只是让预览播给你看，不等于把它带进导出文件。



导出弹层的六项：文件名/格式（GLB）/纹理分辨率（1k）/导出骨骼（开）/动画数量（必须手动调成1，默认是0）/原地播放动画（开）。右边弹出的「选择动画」面板显示1/1，「行走」已勾选。底部还有一个「发送至Blender」。角色是19830面/ 11733顶点的T-Pose男性。2026-09-10。

「动画数量」那个数字从0变成1，才算真的带上了。旁边的「原地播放动画」也要理解一下：打开之后，动画只管迈腿，位移交给游戏代码管。做游戏角色这个开关必须开，不然动画自带的位移会和你的移动逻辑打架。

导出来的是两个文件，不是一个

还有一个更隐蔽的：我一度以为「导出把动画丢了」，其实是它被分成了两份。做小王子那个角色时，点导出的同时抓了一下网络请求，在一串 `studio/progress` 和 `project/detail` 里看见了一个从没出现过的文件名。

文件	里面是什么	体积
<code>tripo_rigging_<uuid>.glb</code>	网格+ 41关节骨架，没有动画	4.81 MiB
<code>tripo_retarget_<uuid>.glb</code>	骨架+ <code>preset:biped:walk</code> （这一件126条通道），没有网格	132 KiB

这两个体积是文件落盘后数字字节数换算的，不是界面上的读数：5,040,200字节和135,540字节，按1 MiB = 1,048,576字节折成4.81 MiB和132 KiB。本节的模型体积一律按MiB写，\$ 22和附录A那张资产表引的是同一批数。

两个文件同源、41个关节同名，用法就是把前者的骨架装进场景、用后者的剪辑驱动，`AnimationMixer`是按节点名字绑轨道的，对得上。顺带解掉了另一个疑虑：既然动画在独立文件里，网格那个文件的静止姿态就是干净的绑定姿势，之前看着像「停在步态上」是我看错了。

还有一条：`tripo_retarget_` 这个文件没有 `_meshopt` 后缀，也就是没压缩，可以离线直接读；带 `_meshopt` 的那些用手写解析器读出来是乱码。

注意

套完动画之后，绑骨页上的「导出」按钮会从DOM里消失。09-12那次我找了半天没找到它。别在界面上死磕，这两个文件的地址在详情接口里本来就有：`GET /v2/studio/project/detail/v3/<uuid>`，返回里 `data.operator.rigging.model_url` 是网格加骨架那份，`data.operator.retarget[0].model_url` 是骨架加动画那份。`retarget` 是个数组，每个元素自带地址。走接口反而是这一步唯一稳的通路。

第一件P2.0绑骨件

前面讲的三件绑骨资产用的都是高精度模式（v3.1）。2026-09-12我用智能网格（P2.0）也绑了一件，这是这本书里唯一的P2.0绑骨样本，数据单独列一下：

项	读数
成品	<code>pedestrian-01-walk.glb</code> ，16,178面/ 1个skin / 1024 ² 贴图/ 1.30 MiB
输入姿态	A-Pose（手臂下斜30到40度）
绑定结果	19秒，第一次就成
骨架落盘校验	41个Cluster + 1个Skin + 82个LimbNode
动画	<code>preset:biped:walk</code> ，单独一个FBX，0.68 MiB

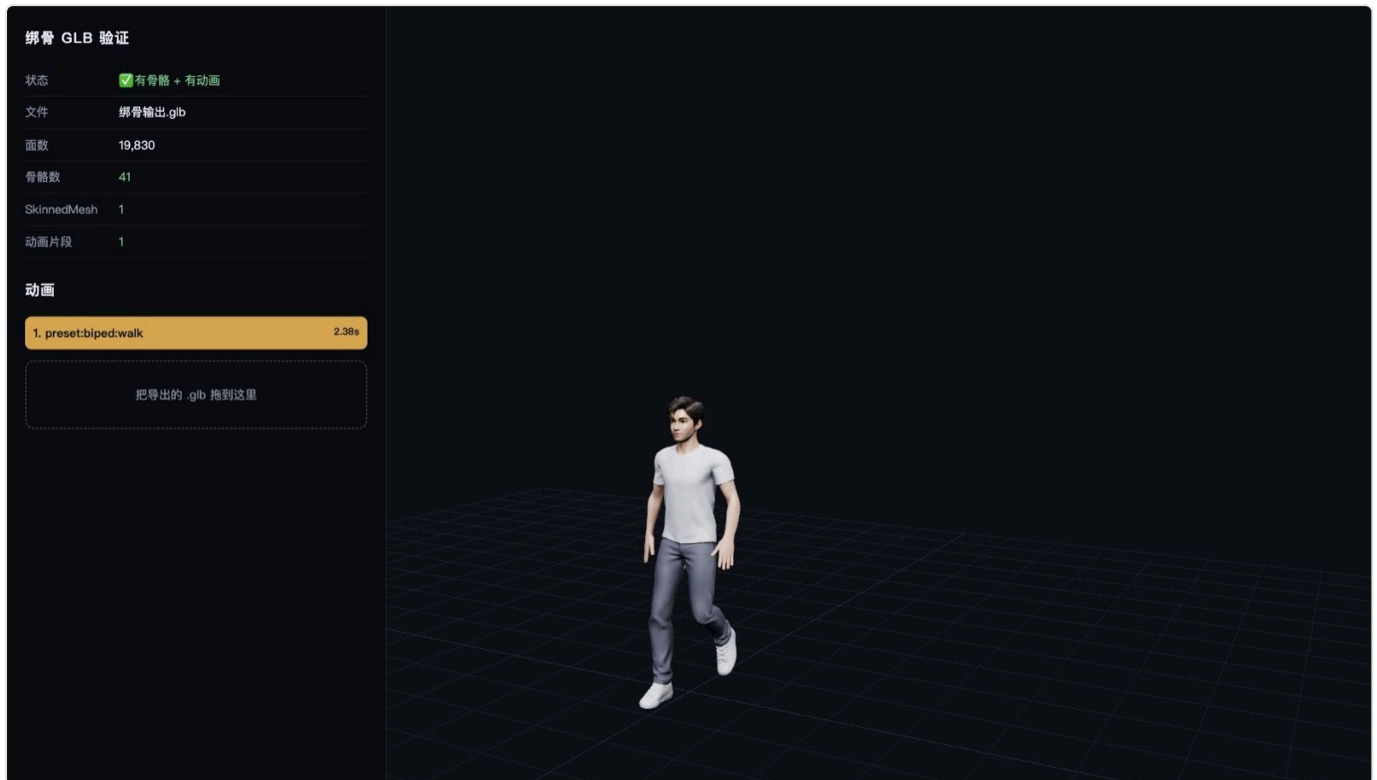
「Cluster」是FBX里存蒙皮权重的那个结构，一根骨头一个，所以41这个数和前面v3.1那副骨架是同一个规模；LimbNode是骨骼节点，82个包含了骨架末端那些不带权重的节点。这些数不是从界面上抄的，是把FBX下来用 `strings` 直接数出来的，凡是能落盘再数一遍的，就别信界面上的数字。

有两件事要提前知道，不然装进项目里会踩：

- 1.这件的导出里带了两条动画，按下标取会取错。名字分别是 `Armature.001|Armature|ArmatureAction` 和 `Armature.001|Armature|preset:biped:walk`，第一条是空的。代码里写 `animations[0]` 会拿到那条空的，人就站着不动。改成按名字挑：`g.animations.find(a => /walk/i.test(a.name)) || g.animations[0]`。之前的资产只有一条动画，所以这个坑一直没露头。
- 2.它的贴图只有一张base color。智能网格出的件就是这个结构，没有normal、没有ORM，所以同一盏灯下它会比高精度那批哑一点。想要金属和粗糙度那两层，要么用高精度模式生成，要么自己在引擎里补材质参数。

验一遍再进引擎

拿到GLB先别急着接，写个二十行的页面把它拖进去数一遍。



自建的验证页读数：状态「有骨骼+有动画」、面数19,830、骨骼数41、SkinnedMesh 1、动画片段1，动画名 preset:biped:walk、时长2.38秒。右边角色正迈开腿，不是滑行。页面源码在 绑骨实测/验证页.html，2026-09-10。

巴黎那个行人的动画数据拆开是这样：**123个通道，其中16个通道带关键帧**（真正在动的腿、臂、脊柱，帧数51到57不等），采样间隔0.042秒（约24 fps），时长2.375秒一个完整走路循环；剩下107个是常量通道，对应手指这类保持不动的关节，属于正常结构。面数从头到尾没涨，19830进、19830出。

这里的123和前面那张两文件表里的126不是同一件资产。但别拿123去认角色——123只是 41×3 这个算式的结果，凡是41关节、只有关节带通道的件都是123（巴黎这个西装行人是，§ 15那个T-Pose男性也是）。126那件是小王子。两件都是41关节，差的三条来自骨架根节点——通道数是「带通道的节点数 $\times 3$ 」（位移 / 旋转 / 缩放各一条），巴黎那件只有41根关节带通道（ $41 \times 3 = 123$ ），小王子那件连骨架根节点也带上了（ $42 \times 3 = 126$ ）。所以看到自己的件数落在这个区间，不用怀疑绑骨出错。

接进引擎的三条硬约束

1.用 `SkeletonUtils.clone()`，不要用普通的 `clone()`。带skeleton的模型普通克隆会共享骨骼，你放一排行人，他们会像仪仗队一样同一帧抬同一条腿。这个工具在 `three/addons/utils/SkeletonUtils.js`，很多精简过的vendor目录里没有它，import会让整个页面白屏。

2.每个实例一个 `AnimationMixer`，相位要错开。我用的是 $(x \times 0.37 + z \times 0.11) \times \text{duration}$ ，拿位置当种子，免得再存一份随机数。

3.把位移轨切掉，只留旋转轨。retarget出来的剪辑里 Hip.position 的Y位移范围是1.338，而人本身才1米出头。行走是代码推的，动画再叠一层位移，循环回到起点整个人会往回弹。我在画面里看到的「走着走着又会后退一大步」就是它。步态本来就是纯旋转的事。

还有一条让角色看起来不呆的小设置： mixer.timeScale 跟着实际移动速度走，走多快推多快。我按每秒2.5米定标，站住的时候把它冻住。不做这件事的话，角色会出现「脚在原地小碎步、人却在飞速平移」那种滑冰感，观众一眼就看出来不对。

朝向要分三类记，别并成一条

「模型进场景之后是背对着你的」这件事我处理过太多次，一开始我以为有一个统一的角度能兜住全部，后来发现朝向惯例是按来路分的，有三套，混着记必然装反。

这一类模型	自身正面朝哪	怎么处理
Tripo绑骨预设出来的人形	不是+Z，实测差90°	小王子那边写死了 PRINCE_YAW = -Math.PI/2
高精度 (v3.1) 那批静态资产	行人+Z、建筑-Z（宽立面在±Z，窄面在±X)	同一批资产里就有两套惯例，要分开定
智能网格 (P2.0) 那一族	+X，38件里36件如此	去顶替旧行人时统一 yaw = -90，顶替旧建筑是 yaw = +90

第三行那两个例外没法靠推理找出来，只能渲图看：一辆车的正面在-X（+X那一面是尾灯和后窗），一件靠墙的书箱正面在-Z（-Z那一面书箱是敞开的，+Z是合上的箱盖）。

注意
四视图分不出正面和背面，必须渲六视图。常见的四视图是-Z（前） / +X（右） / +Y（顶） / 斜45°，人形的正面和背面在这四个机位里长得太像，尤其是戴帽子穿大衣的角色。补上+Z和-X这两个机位再看，「在哪个视图里看见正脸，正面就朝那个方向」，一眼就定了。包围盒帮不了你： yaw 90 和 yaw -90 的包围盒一模一样。



六视图对照，机位从左到右是-Z / +X / +Z / -X / 顶/斜45°。上行是v3.1的旧行人，正脸出现在第三格（+Z）；下行是P2.0的新行人，正脸出现在第二格（+X）。两件包围盒读数差不多，只有这张图能把朝向区分开。渲染脚本 `_verify/render_glb6.py`，全本地跑，不开浏览器，2026-09-12。

量一个绑骨模型的身高，比你想象的难

这条我栽得很惨，写出来给你省两轮。我要把小王子和那张1米01的书桌摆在一起，得先知道他多高，于是写了个函数去量包围盒。数收敛了，拟合也漂亮，量出来的结论是1.552米。这个1.552是测量假象，真实渲染出来的人只有0.79米，站在书桌旁边比桌面还矮。

往下看之前先把三个长得很像的数摆清楚，不然后面每一句都容易读岔：

这个数	数值	它是什么
目标值	1.55米	代码里设死的那个高度（ <code>normalize()</code> 的 <code>target</code> 、身高探针的 <code>want</code> ，用的都是它）。这个数从头到尾都是对的
测量值	1.552米	坏掉的量法给出的读数。它离目标值只差0.002，看着像「拟合成功了」，其实是假象
实际渲染值	0.79米	屏幕上真正那个人的高度。和1米01的书桌摆进同一张图才露馅

错的不是1.55这个目标值，是量法。底下三处问题叠在一起，每一处单独看都不显眼；三处都修完之后，身高探针一次收敛到1.547米，正好落回目标上：

症状	正解
量的是世界Y的极值。建世界时角色还在原点、旋转是单位阵，世界Y恰好就是人的上轴，量出1.55——这一下量到的正是 <code>normalize()</code> 规整出来的那个目标高度；一摆到球面上人就是斜的，同一副骨架量出0.89	沿角色自己的上轴投影。而且上轴要取带朝向的那个节点，取错了「沿上轴量」和「沿世界Y量」会给出一模一样的结果

症状	正解
<code>bindMatrixInverse</code> 是陈旧的。 <code>three.js</code> 只在 <code>updateMatrixWorld()</code> 里刷新它，而 <code>SkinnedMesh</code> 重写的正是这个方法，不是 <code>updateWorldMatrix</code>	先 <code>updateWorldMatrix(true,true)</code> 刷祖先链，再 <code>updateMatrixWorld(true)</code>
上面两条叠加，让「高度对缩放」呈现出一条带截距的直线，于是很自然地去解那条直线	量法修对之后关系本来就是对的，不需要拟合

为什么会连着两次判错：第一版只验证了「数收敛了」，没验证「画面对了」。这类bug可怕在它全链条自证，拟合收敛、代码不报错、模型和骨架都是好的，连「贴地0.064米」这种细节都算得出来。

量骨骼网格的时候，量法本身就是被测对象的一部分。换个人、换个角度、换个时机量，答案就变，那说明量法还没对，不是精度不够。最省事的验法是把它和一个已知尺寸的东西放进同一张图里看一眼。

一个能走的角色，花在哪几笔

动作	积分
生一张参考图	0（走的是订阅额度，不花Tripo积分）
Tripo图生3D（高精度，默认8K贴图档）	50
模型是账号内已有的，不用上传	0（走上传是50导入费）
自动绑定	20
套动画	0
导出	0（实测余额前后不变）
合计	70积分

以上是截至2026-09的官方标价，以官网为准。按¥0.0467一分算，一个会走路的角色约¥3.27。这70分是我09-10那次逐笔记下来的，那一件没有重试；要是碰上一次重绑，这一件就是90分、约¥4.20。

这套流程后来在巴黎那座城上又跑了一遍，主角换成另一个行人角色（`pedestrian-01`），绑法完全一样，只是它那次第一遍没绑成，多花了20分。街上18个行人里有6个由那一个绑骨模型实例化出来、真的在走，其余12个是五种不同的静态行人站着（一开始是18个全用它，那其实是个bug，根因见 § 18），桌面全流程和三种移动端尺寸的回归测试都通过。两个角色是两条线上的两件事，账也分开记，别把它们当成同一个。

核心建议

批量做角色，先按件数算一遍再动手，这个数很容易低估。拿游乐园那个demo当尺子：园子里有44个行人，假如每一个都在Tripo里单独生成再绑骨，按一件70分算就是3080积分，比Pro一个月的3000还多——那批行人实际是在代码里搭的骨架，这3080是照这条流程折算出来的，不是我真花掉的钱。更划算的做法是做3到5个不同外观，其余靠实例化加随机相位错开，观众分不出来，积分从三千多降到三百上下。

还有一条我没测的，得说清楚：近一百个动画里我只实地验过walk这一个，跑步、挥手、舞蹈的质量稳不稳定，我不知道。面部和口型那一层也完全没碰，所以这本书里不会出现关于它的任何结论。

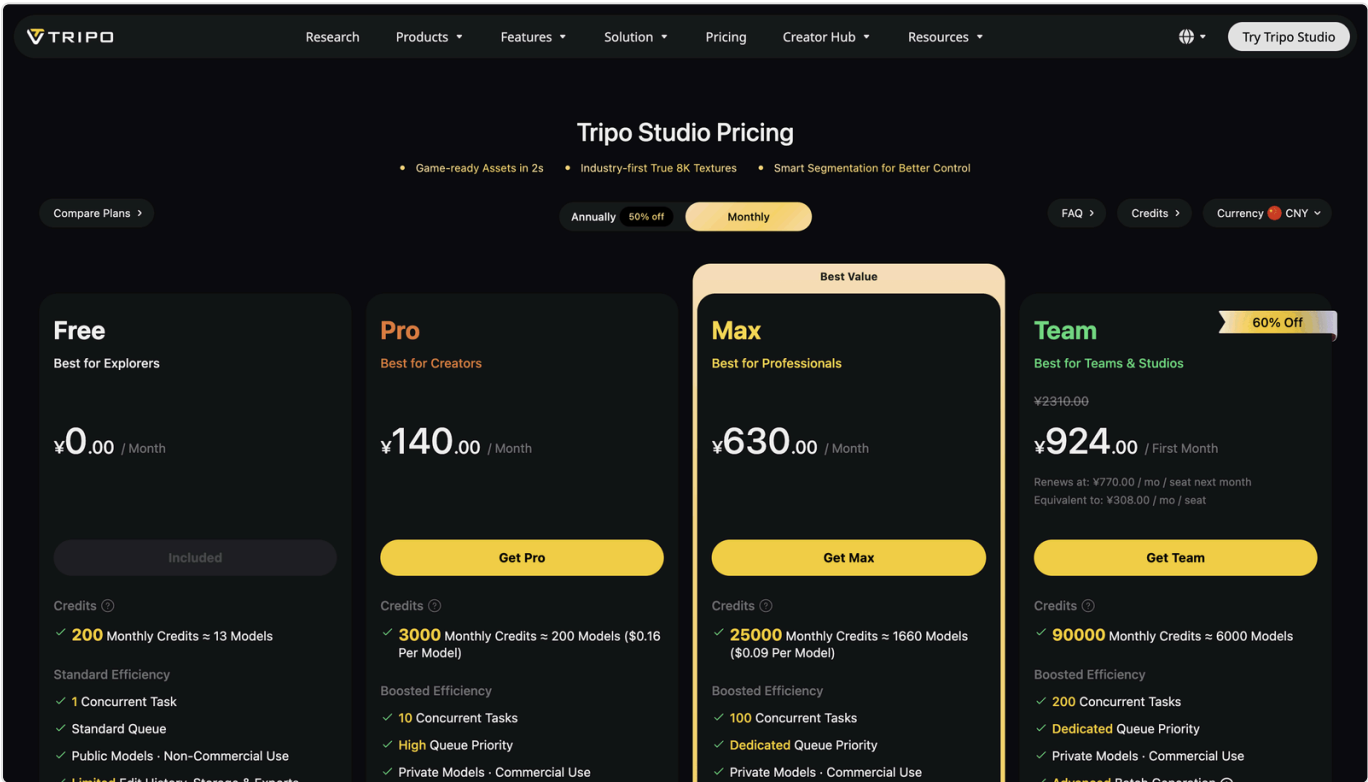
§07 怎么把积分花在刀刃上

Spending Credits Where They Count

这一节讲的是配置选择：每个按钮上那个数字是怎么构成的、同一件资产有几种配法、什么时候该配高什么时候该配低。学会读按钮上那个实时变化的数，比背价目表有用得多。

这一节里所有价格都是截至2026-09的官方标价，以官网为准。我把采集日期写在每张表底下，是因为这类数字本来就会变，你对账时以当天页面上的为准。

订阅这一层：四档，差别不只是积分数



官方Pricing页，右上角Currency切成CNY之后的读数。四档：Free ¥0、Pro ¥140/月、Max ¥630/月、Team ¥770/席/月（页面当天挂着60%off的首月促销价¥924，次月起按席收）。Team那个总价和单席价差三倍，是3席起售推出来的——月付\$110一席、\$330总价，年付\$55一席、\$1980总价，两条各自独立的算式都得3，而页面上从头到尾没写过这个最低席数。页面顶部三句标语：Game-ready Assets in 2s / Industry-first True 8K Textures / Smart Segmentation for Better Control。2026-09-11。

	Free	Pro	Max	Team
月费（CNY）	¥0	¥140	¥630	¥2310（3席起）
月积分	200	3000	25000	90000
官方折算	≈13 models	≈200 models	≈1660 models	≈6000 models

	Free	Pro	Max	Team
并发任务	1	10	100	200
模型归属	Public · 非商用	Private · 商用	Private · 商用	Private · 商用

截至2026-09官方标价，以官网为准。

最后那一行比积分重要得多。**Free档生成的模型是公开的，而且标着非商用**；从Pro开始才是私有加商用。你要是打算把做出来的东西放进产品里，这一行就是门槛，不是积分够不够的问题。

免费档能做到哪一步

200积分按基础价15算是13个模型，按界面默认的50算是4个。这个量足够你把整条链走一遍，看看生成质量对不对得上你的需求，但不够做一个项目。

真正卡住你的也不是数量，是另外两条属性：模型归属，和导出。归属那条上面刚说过，导出这条我没能验到底。

「**免费档到底能不能导出**」，我手上两个来源强度不一样，都摆在这儿，这也是全书这件事的统一口径。官方 Compare Plans对照表里，Free那一列 Exports 的原文是 15 (H2.5 only) —— 额度有15次，但限定在H2.5这一档模型上；同一列的Private Models和Commercial Use都是❌。另一边，我自己的项目记录里写着GLB / FBX这些格式全部导不出、必须订阅，**但我从Pro起步的，这一条没在免费账号上复现过，也没有截图**。对照表原文、逐行读数，以及H2.5到底是哪一档模型，都在附录C。

这个洞我没补上，但它不挡你做决定，所以先把结论放在这儿。「要不要付钱」这个决定卡在模型归属那一行，不卡在导出这一行。导得出也好、导不出也好，免费档的产出都是公开的、标着非商用。所以免费档只该用来回答一个问题：它生成出来的东西，够不够你用。够用，再考虑从Pro起步；不够用，你连这一步都省了。

至于导出本身，你不用等我，注册完三分钟就能当场问清楚：生成一件，点Export，把你真正要的那个格式（多半是GLB或FBX）点到底，看它让不让你下载。导出**不扣积分**（本节后面有我拿余额验的那一笔），所以这一步花的是三分钟，不是那200分里的任何一分。**要点的是你真正要的那个格式，别只点一个能导的就当过了**——别等做完一批资产才发现拿不走。

核心建议

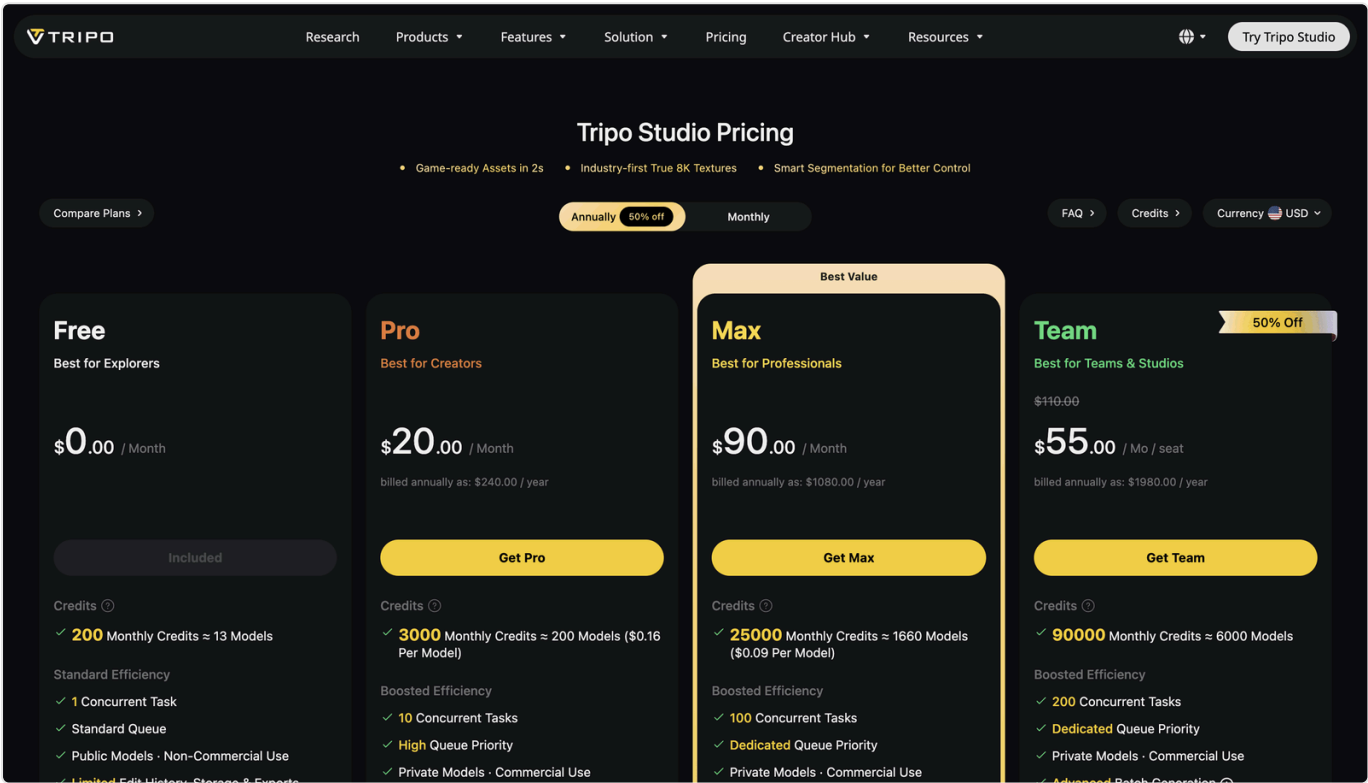
在你自己点出结论之前，预算按这一条来最稳：**免费档能试质量、不能交付**。两个来源哪个准，我没在免费账号上复现过，不替你裁；但它们指向的是同一个做法——导得出，那15次也限定在H2.5这一档，而且同一列的Private Models和Commercial Use都是❌，文件拿到手也不能放进产品；导不出就更不用说了。**所以免费档的位置是「验质量的试用期」，别把它排进交付环节的预算，到要交东西那天会卡住**。真要交付，从Pro起步；至于导出这个洞，开工第一天花三分钟按上面那三步点一遍，你自己账号上当天就补上了。

并发数也是个隐形差别。Free是1个并发，意味着你提交一件就得等一件；Pro是10个。做一批二十几件资产的时候，这个差别比积分更影响体感。

一积分值多少钱

¥140除以3000，一积分约¥0.0467。这本书里所有的人民币数字都按这个价折算。

这个基准价我复核过，因为它撑着全书所有的¥数字。2026-09-13不带账号、不带登录态重取了一次官网Pricing页，Compare Plans的Price行还是Pro \$20 /月、\$240 /年，和带着优惠券采集那次一模一样。另一条旁证是三个付费档的CNY对USD比值：140÷20、630÷90、2310÷330全都正好是7.0——优惠券只会打歪其中一格，不会让三格一起保持同一个汇率。采集当天我账号上确实挂着一张首月券，它折到的是卡片顶部那个大售价，没折到对照表，验法写在附录C。

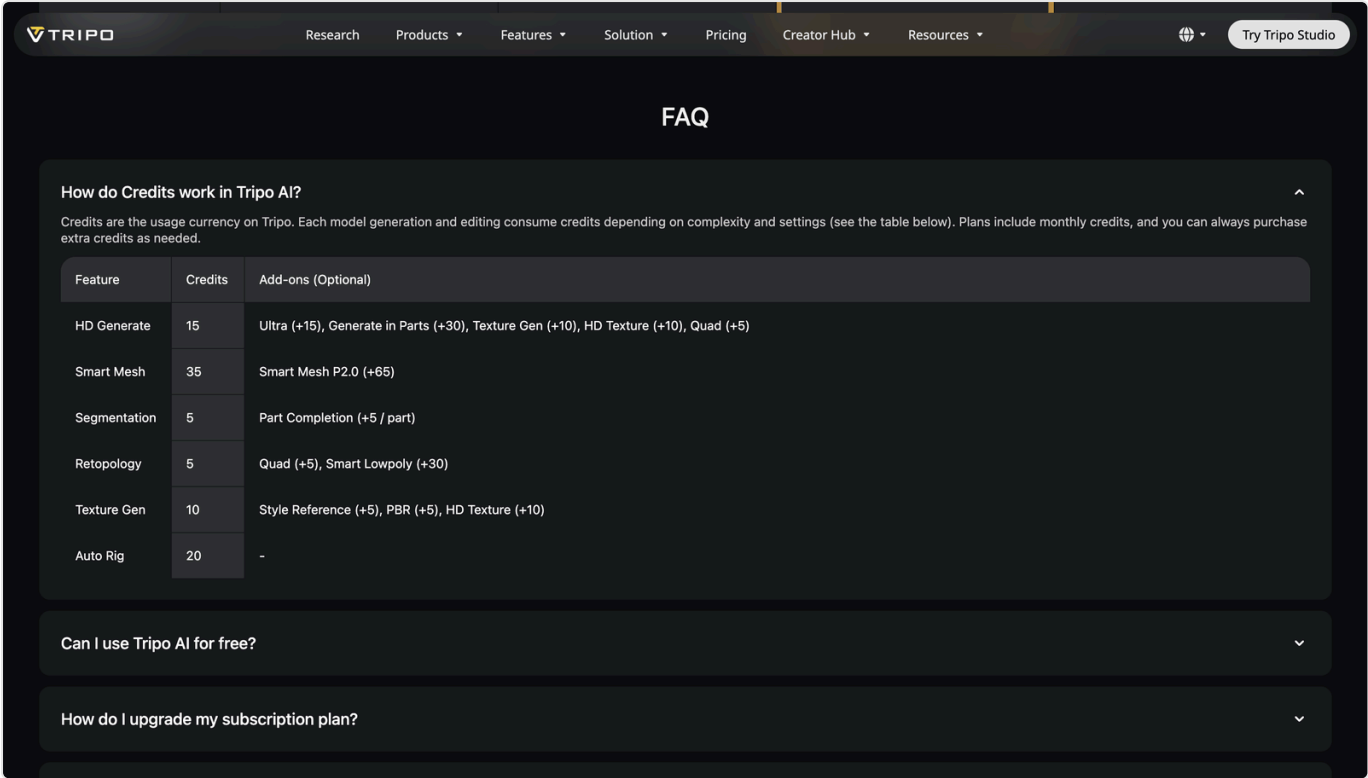


同一个页面切到Annually那一档，标签上挂着「50% off」，Pro显示\$20/月、按年计\$240。促销状态天天在变，我抓接口那天Professional年付折算下来是每月\$13（标35% off）。对外讲成本要用官方标价，不要用自己当时抢到的促销价，读者是按标价做决策的。2026-09-11。

我账号里那两万多的余额不是订阅送的，是另外买的积分包。这个也得说清楚，不然你看到截图里的余额25220，会误以为Pro一个月给这么多。

官方那张表，和界面上量出来的那一套

官网Pricing页的FAQ里有一张积分表，叫「How do Credits work in Tripo AI?」，把每个动作的基础价和可选加项分开列。这是官方唯一一处把这件事说明白的地方，生成按钮上不会告诉你。



tripo3d.com/pricing页面底部FAQ第一条展开后的官方积分表。官方表是三列：Feature / Credits / Add-ons (Optional)，每一行的加项只属于这一行的功能。六行基础功能是HD Generate 15、Smart Mesh 35、Segmentation 5、Retopology 5、Texture Gen 10、Auto Rig 20，其中Auto Rig那一格官方写的是「-」，没有加项。**HD Generate那一行的15，就是生成一个模型的真实底价。**2026-09-11 14:50。

官方那张六行的积分表（每个功能的基础分和可选加项）在附录C，截至2026-09的标价，以官网为准。

拿它去对界面上实测到的数字，两项严丝合缝，两项要多看一眼：

拿它去对界面上实测到的数字，两项严丝合缝，两项要多看一眼：

界面显示	拆开	对得上吗
生成50	按官方表只能凑到15 + 10 + 10 = 35，还差15。把Ultra的+15算进来才够50，可默认状态下Ultra Mesh Quality是关着的	×总数对得上，但那个拆法和界面默认状态不一致
重拓扑10	Retopology 5 + Quad 5	✓
绑骨20	Auto Rig 20	✓
智能网格「100划掉65」	官方表的Smart Mesh 35 + P2.0 65 = 100，对上的是被划掉那个原价；实扣的是折后的65	△原价对得上，实付要看划线后那个数

第一行不能靠凑。默认表单里Ultra是关的，所以「15 + 10 + 10 + Ultra 15」这个拆法虽然凑得出50，描述的却不是你打开工作台时看到的那个状态；照着它去界面上找那15，会把本来关着的Ultra打开，一次多花15积分。我在工作台里把开关一个一个关过去量了一遍（过程和截图在 § 01），量出来的是另一套结构：几何15是底价，贴图是打包卖的，2K加15、4K加25、8K加35，Ultra Mesh Quality单独再加15。

把同一个表单从「只要几何」一路开到「8K + Ultra」，按钮上的数是15 / 30 / 40 / 50 / 65五档，逐档对照表在附录C。

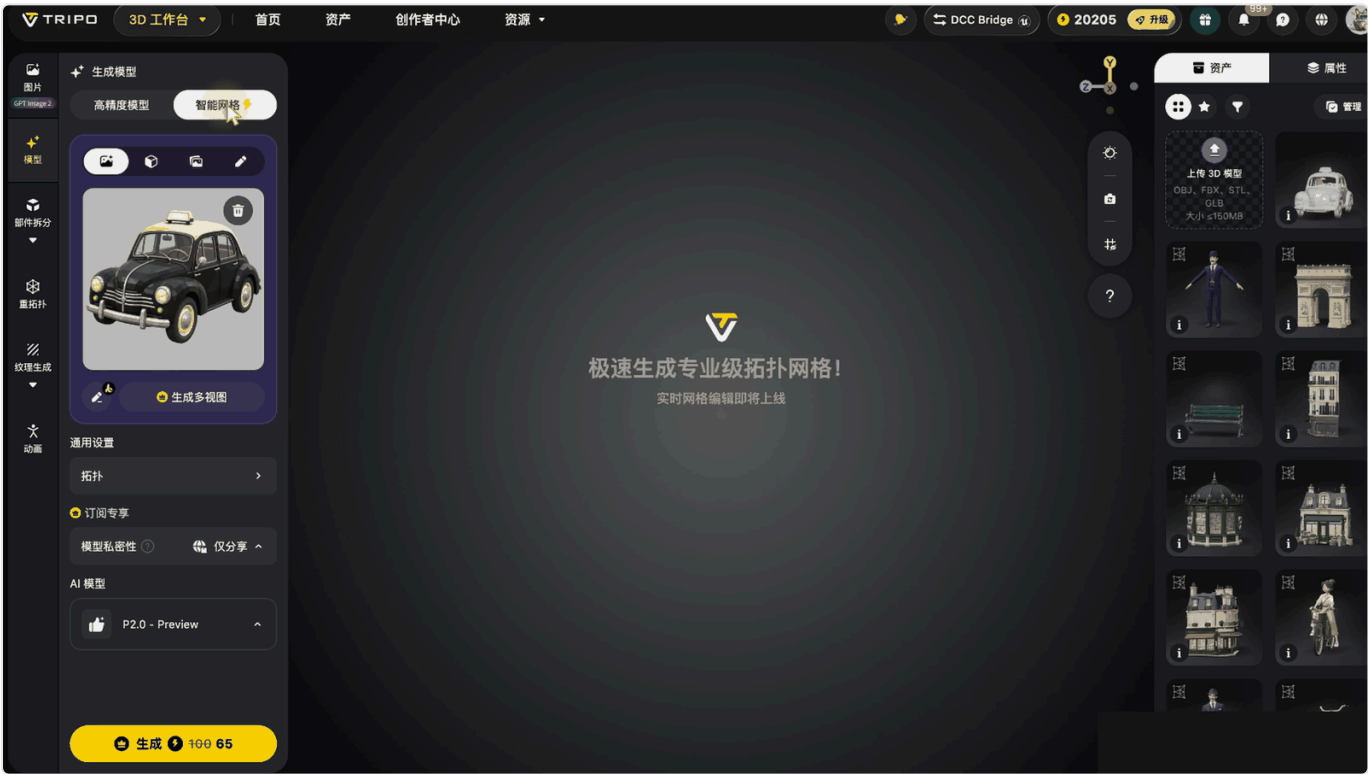
截至2026-09界面实测，以当天按钮上的数为准。

两套分法的差别很实在，它决定你能动哪一块。界面把Texture Gen和HD Texture打包在一个开关后面，你没法只关其中一项；能动的是贴图档位，从8K拨到2K少20分。而官方表里那个15分的Ultra，在工作台的默认状态下本来就是关着的。

所以那个50不是几何本身的价钱，其中35分买的是8K贴图。这和面数那边是同一件事的两面：默认值替你做了选择。差别在于面数那个默认值（高精度模型那边的面数控制默认20000，而滑块量程是500到200万）恰好落在可用区间，而贴图默认给的是最贵的8K档。知道了这一点，你就有了一个可以调的旋钮。

智能网格那条线实扣多少

这条线我对得最细，因为09-12那天我连着跑了十几笔，每笔都拿余额去核。



生成表单切到「智能网格」，AI模型是「P2.0 - Preview」，底部黄色按钮写的是「生成 100 65」：100被划掉，实扣65。顶栏余额20205。这一档的贴图不含在里面，出来的是灰模。Tripo Studio中文界面，2026-09-12。

这一步	界面	实扣
智能网格生成（P2.0 - Preview）	「100划掉65」	65
生成完再走一次贴图生成	「纹理生成30」	30
一件带贴图的完整成品	—	95

截至2026-09界面实测，以当天按钮上的数为准。

怎么验的：那天主批12件网格是 $12 \times 65 = 780$ ，另有一件重做和一件绑骨用的行人，网格共14笔910；加上13笔纹理的 $13 \times 30 = 390$ ，再加一笔绑骨20，合计1320，和当天余额从21540掉到20220这一段一分不差。逐笔对得上，这个65就不是我看花了眼。

核心建议

选高精度还是选智能网格，真实的差价是15对65，不是50对65。两边都只要纯几何的话：高精度把Texture开关关掉是15，智能网格是65。这两个数才是同一口径下的对比。要是两边都带贴图，那就是50对95。先想清楚你要的是哪一种几何，再看价，顺序反了很容易选错档。两种模式各自适合什么，在 § 04 有一次同图同面数的对照实验。

官方自己的算法验证了「基础价15」

对完账我心里还是有点不踏实，因为FAQ那张表是我抄下来的，万一抄错了呢。后来发现定价页上有一个可以反推的东西：每一档都写着「N Monthly Credits $\approx$ M Models」，拿N除以M就行。

套餐	月积分	官方折算	积分 $\div$ 模型数
Free	200	≈ 13 models	15.4
Pro	3000	≈ 200 models	15.0
Max	25000	≈ 1660 models	15.1
Team	90000	≈ 6000 models	15.0

四档全部落在15上。官方在算「你的积分能做几个模型」的时候，用的分母就是15，不是50。这是产品自己在页面上确认了基础价。

这条对你的实际意义是：**Pro那3000积分不是只够做60个模型，是够做200个**。前提得说清楚，官方那个分母15算的是纯几何，不含贴图；真要带贴图，2K档一件30分，3000积分是100件。就算按这个算，也比按默认的50分一件多出一倍。

工作台高精度模型表单的全貌在 § 01：通用设置下是「几何与贴图」，订阅专享区有「分部件生成（带New角标，默认关）」。

但按钮上的数字才是当场生效的那个

我得把一处对不上的地方摊出来。首页那个快捷生成框里有个二选一，高精度模型和智能网格，选前者时按钮写的是55，不是50。

首页那个快捷入口的样子见 § 01：拖一张图进来，下面高精度模型/智能网格二选一。**选高精度模型时按钮是「Generate 55」，比工作台表单的50多5分——这5分对应哪个开关，我没在界面上核出来。**

同一个框切到智能网格（图见 § 01），按钮变成「Generate 100 0」，走的是智能网格那条线，Preview期间第一次免费。同一张图、同一个框，两个按钮的数差着一倍，差别只在上面那个二选一。

别背价目表，按下去之前看一眼按钮上那个数就行。它跟着你当前的开关实时变，而且不同入口的默认开关不一样，还会叠上当期的划线折扣。想知道某个加项值多少分，把它关掉，看那个数字掉了多少，这比查表准。

Trial x1：免费那一次之后，落在折后价上

界面上有好几处挂着「Trial x1」或「New Function Trial」的角标，意思是这个功能给你免费试一次。我撞上的有这些：

功能	角标	页面原价	实付
智能网格P2.0	Preview Trial x1	100	首次0，之后每笔65（原价被划掉的折后价）
智能拆分	New Function Trial x1	40	0（那天）
分部件生成	Trial x1	+30	没试

截至2026-09界面实测，以当天按钮上的数为准。

有一处要说在这儿，免得你翻到 § 01时犯嘀咕：上表「分部件生成」那一行记的是我2026-09-11在国际站界面上看到的角标；§ 01那张09-14拍的国区截图上，这一项挂的已经是New，不是Trial x1。是国区和国际站的差别，还是这三天官方改了角标，我没分开验。角标会变、价目表会过期，只有按钮上当场显示的那个数不会——这正是全书反复说「以你按下去之前按钮上那个数为准」的原因。

智能网格的AI Model下拉见 § 01：P2.0 – Preview带「Trial x1」，另一项是P1.0 – Fast。规划预算时按你实际会付的那个数算，Trial只有一次。

这里有个我早期算错过的地方：我一度以为Trial用完就回到原价100，做批量预算时按100乘件数算，结果多留了一大截。真正扣的是划线后的65，一批12件就差420分。预算宁可按看到的按钮数算，别按记忆里的价目表算。

余额在哪看

Assets页的样子见 § 03。这里只提一件事：积分余额常驻顶栏右侧，旁边就是Upgrade；下面那排All / Smart Mesh / Untextured / Textured / Rigged筛选，做完一批资产用它来点数最快。

余额的变动可以当账本用。那辆P2.0出租车做完贴图，余额从25220变成25190，正好30分，和按钮上写的对得上。每做一步记一次余额，比事后回忆准得多。完整的一串读数在附录C，四个时间点互相闭合。

按用途选配置：同一件资产有五种配法

讲完构成，说真正要做的决定。一批资产不该一个配置跑到底，该先给每件定个用途，再按用途选档。这是我做完三批资产之后才固定下来的做法。

同一件资产有五种配法，单价从15到65——差的几乎全在贴图档位上：主角走8K（50）、中景4K（40）、远景2K（30）、只要形状就把Texture整个关掉（15）、要干净四边面走智能网格P2.0（65）。五行完整对照表在附录C。

巴黎那批23件我走的是4K那一档，一件40；小王子那16件是默认的8K，一件50。而在画面里，一件只在两百米外出现的建筑，8K和2K我分不出来。

巴黎那批23件我走的是4K那一档，一件40；小王子那16件是默认的8K，一件50。同一个「高精度模型」两个单价，差的全在贴图档位上，而在画面里，一件只在两百米外出现的建筑，8K和2K我分不出来。

核心建议

判断哪一档的土办法：把这件资产在你的画面里最近能被看到的那个距离，先用现有资产摆一遍试试。看得清砖缝就上高档，看不清就往下拨。凭想象定档，十次有八次定高了。

核心建议

导出不扣积分，这一条我拿余额验过。09-12那天导一份FBX下来，导出前后顶栏余额都是20205，一分没动。所以别为了省积分而不敢多导几次，同一个模型换几种格式、换几档纹理分辨率反复导，成本是零。另一头有笔钱要记住：把一个外部模型上传进Tripo做绑骨，要50导入费，而从账号内已有的资产里直接选是0，§06有这条路的完整走法。

这些积分花在哪了

八条线逐笔花在哪，那张流水表在附录C（含每一笔的用途和失败重试）。小计1350积分，约¥63。

注意

这张表的口径，有三样东西不在里面。一是巴黎那条线9月9日之前的生成，二是Unity那一版（记在它自己的文档里），三是9月11日下午那批38件的P2.0批量，一次3640积分，单独记在附录C的第二段流水里。第三样的量比这张表还大，之所以分开记，是因为它从一开始就是拿余额读数记的账，和这张表「按各条线记录加总」不是同一种记法，混在一起会把两种可信度不同的数搅成一个。两段的完整流水、以及一笔没对上的10分，都摊在附录C。

要一个最硬的数的话，用这个：从9月11日那个25220的读数，到9月12日收工时的20120，Tripo这边净掉了5100积分，按¥0.0467一分算约¥238。这是余额头尾相减，不靠任何一条记录加总，也是这本书里我最愿意拿出来对外说的一个成本数字。上面那张1350的表发生在25220这个读数之前，属于另一段，两个数不要相加。

这张表里有两个数字值得单独看。一个是小王子那条线的840分，占了六成以上，因为那是唯一一条「按设计图清单批量生成」的线，16张图各生成一次。真正吃积分的从来不是难度，是件数。另一个是星月夜那200分里的

50，那是我误判一次生成失败、重复提交造成的，属于纯浪费。

顺着这个往下想，钱该花在哪就清楚了。件数是省不掉的，你要几件就是几件。**能动的只有每件的配置档位，和那些白交的重复提交。**把远景件的贴图档位拨到2K，单价从50降到30；提交前先看清楚状态，别在任务还在跑的时候以为它失败了。这两件事加起来，同样这些产出大概能落在800分上下。

另一笔账：时间

积分好算，时间容易被忽略。我记下来的几个数：

动作	耗时	花积分吗
高精度模型生成一件	端到端约四分钟（表单上标着Expect longer wait times；这一档没有逐件算子耗时，口径见 § 05）	50
智能网格生成一件	17到65秒，随面数预算变	65
贴图生成一件	109秒	30
自动绑定一次	3到19秒	20
本地批量减面25件	131秒 ，平均5.2秒/件	0
19分钟做完一个可玩demo	星月夜那条线的实际用时	200

上表里Tripo侧那几个数是**算子耗时**（算子耗时口径，不含上传与排队；端到端要往上加，两个口径的实测差见 § 01，词条见附录D）。**排期按端到端算，别按接口报的算。**

这张表里有个挺反直觉的东西：**最花钱的那几步，你根本不用盯着**，提交完就能走开。真正吃你注意力的是后面组装和调试那一大段，而它一分钱不要。所以别一件一件提交，Pro给的10个并发是实打实的产能，一次丢一批出去，回头统一收。

怎么记账

我这一轮之所以能把每笔说清楚，靠的不是事后回忆，是两件很笨的事：每条线在自己的记录文件里留一行「这条花了多少积分、花在哪」，以及关键步骤前后各看一眼顶栏余额。

这两件事的可信度不一样，我是吃了亏才分清的。**余额掉了，是扣费真的发生过的正面证据；记录加总只能证明「我记下来的那些确实花了」，证明不了「没记下来的不存在」。**所以两边打架的时候以余额为准，然后老老实实把差额写出来。我09-10那一段就是只有加总没有余额，现在回头看，账面上的洞正好都在那一段。

余额差法特别好用，因为它不受界面显示影响。那辆出租车做贴图，余额25220变25190，就是30，跟按钮上写的一致；09-12那12笔网格，780分也是这么逐笔对上的。要是哪次对不上，说明你理解错了某个开关。**产品说**

的价和实际扣的分，得自己对一遍才算数。

还有一处没对上，如实写在这儿：官方表里Segmentation是5、每部件再加5，而Studio的拆件按钮上明晃晃写着40。我只拆过一次（09-11那辆出租车），那天按钮挂着Trial、实扣0，没机会用余额差去验。所以本书凡是提到拆件40，说的都是界面按钮上的数，这一条我还没验完。你自己一次拆件就能验完——两个前提和完整说明在附录C。

还有一条成本线我没走

整本书讲的都是网页订阅这条路。Tripo另有API和CLI，走那条的计费方式、额度形态和网页端不一样，我这一轮没有实测，所以这一节里的任何数字都不要往API上套。那部分在 § 11，等真正接过一遍再写。

把预算铺开的几条做法

做法	效果	适用条件
按用途给贴图定档，远景件拨到2K	单价50 → 30	近景特写留高档，主角别省
只要几何，材质在引擎里给	单价50 → 15	拿到的是灰模。事后再走贴图生成是30，加起来45，所以要一开始就确定
导出时纹理分辨率选1k	不动积分，省的是显存和加载时间	几乎没有代价，游戏距离上看不出
一个几何配N张贴图，而不是生成N个模型	50+25×N，对上50×N	形状全一样，只适合建筑、道具这类
形状对了就改重拓扑，别重新生成	10，对上50	只在形状本身是对的时候成立
角色做3-5个，靠实例化铺开	44个行人，3080 → 300上下（按一件70分折算，见 § 06）	需要错开动画相位，否则像仪仗队
减面和压贴图在本机跑	0积分，5.2秒/个	要装node，命令在 § 04
绑骨用账号内已有的资产，不上传	省50导入费	模型本来就是Tripo生成的

核心建议

开工前先按这张表算一遍总账，再决定订哪一档。一个二三十件资产的小游戏，Pro那3000积分足够，真正的成本大头是你自己的时间。

§08 为什么先出设计图再建模

Design First, Model Second

这一节讲我那两个世界共用的那条流水线：先让编程Agent画一张设计图，再把图送进Tripo变3D。开篇就是那四条参考图配方（看完视频来找它的，就在下一页），然后是中间这一步为什么不是多余的，以及怎么用Tripo自己起的模型名当一道不花钱的验收。

先把那四条配方给你

视频里我念的那四条要求，就是下面这个框。你现在什么都不用懂，把它照抄进你给出图模型的提示词里，拿回来的3D模型质量会立刻不一样。为什么是这四条、每一条各挡掉了什么，§09一条一条拆。

参考图配方·四条，一条都不能少

- ① **一个物体**：画面里只有一个主体，不摆场景、不放第二件东西
- ② **白底**：纯色背景，最多留一点极淡的接触阴影
- ③ **四分之三视角**：一张图里同时交代正面和侧面，Tripo才补得出体积
- ④ **整个东西都在画面里**：顶天立地全进画框，边缘别切掉任何一块

这四条不是我总结出来的口号，是我真跑过的提示词里逐字写着的。下面这段可以整段复制，粘在你自己那句「画面内容：……」的前面或后面：

单个物体、完整入画、居中放在纯白背景上，四分之三视角，只有极淡的接触阴影、无其他物体、无文字、无人物。

参考图配方

让 GPT-image 出的图更好建模的四条要求

写进提示词就行

1 一个物体

画面里只有一个主体，不摆场景、不放第二件东西

提示词原文：单个物体·无其他物体、无文字、无人物



2 白底

纯色背景，最多一点极淡的接触阴影

提示词原文：居中放在纯白背景上·只有极淡的接触阴影



3 四分之三视角

一张图里同时交代正面和侧面，Tripo 才补得出体积

提示词原文：四分之三视角



4 整个东西都在画面里

顶天立地全进画框，边缘别切掉任何一块

提示词原文：完整入画



四条原文出自 调研/实测记录/游乐园-可漫游-20260910/unity-验证/游乐设施/参考图/jobs.jsonl (8 条提示词逐条都写了这四项)

四条配方各自对应的提示词原文，以及一张符合它的真实设计图。四条原文出自游乐园那批的 jobs.jsonl，8 条提示词逐条都写了这四项，一条没漏。

旋转木马

游乐园·游乐设施

先出图，再建模

1 提示词

皮克斯动画电影的道具设定图：

- 单个物体
- 居中放在纯白背景上
- 四分之三视角
- 完整入画

一整座华丽的旋转木马，金色雕花顶棚、红白条纹的圆形底座、中央立柱，木马清晰可辨。

jobs.jsonl 第 1 行

2 设计图·GPT-image



01-旋转木马.png

3 生成结果·Tripo



carousel.glb

一个物体 白底 四分之三视角 整个东西都在画面里

Tripo·图生 3D

四条配方全中的样子：提示词（左）→设计图（中）→Tripo结果（右）。01-旋转木马.png 送进去出来就是 carousel.glb，金色顶棚、红白条纹底座、一圈木马全部还原。注意中间那张图的底是真正的纯白，这是Tripo最省力的情况。

注意

一处得说准的地方：这本书里的例图不全是白底。小王子那16张设计图用的是浅米灰底，巴黎那两批用的是浅灰底，只有游乐园那批是真正的纯白。三批的说法不同，四项要求是同一套。配方第二条的要点是「**纯色、无渐变、和主体明显不同色**」，不是某个具体色号。我用浅米灰是因为纯白容易和物体本身的白色高光区糊在一起。你要是懒得挑，就照原文写纯白，那是最稳的。

小王子那颗星球上摆着16件Tripo模型，巴黎那座城里摆着23件（两个demo的样子见 § 00，展开在 § 22和 § 18）。

两边都没有一件是我手搓的——**我不会写代码，也不会用Blender建模，这本书从头到尾的前提就是这个。**也没有一件是从素材站下的，全部是从零生出来的。

两边走的是同一条路：**Codex先出一张设计图，Tripo再把设计图变成3D。**

文生3D一步到位，坏处是出了事你不知道该改哪头

Tripo的生成表单上有文本页签，你直接打字也能出模型。我一开始也觉得中间那张图是多余的：既然它能听懂「一只坐着的狐狸」，为什么要绕道先画出来？

跑完两批之后我改主意了。三个理由，按我自己认的重要性排。

中间那张图是唯一能让你定位问题的东西。文生3D是一次生成，出来不对，你分不清是自己的描述没说清楚，还是它的形体重建没做好。插一张设计图之后这两件事就分开了：图不对，改文字；图对了而模型不对，那就是重建这一步的事，跟你怎么写没关系。批量做的时候这一条特别值钱：16件资产里有一件翻车，你得在两分钟内知道往哪修。

改图便宜，改模型贵。设计图走的是编程Agent的订阅额度，重出一张不额外花钱；Tripo的v3.1高精度生成一次50积分。我在小王子那批上重出过两张设计图（§ 09里的「短茎大花头」那两张），一分钱没多花。同样的犹豫如果发生在文生3D上，就是100积分。

一批资产的风格统一，只能在2D那一层控制。3D模型的「风格」其实是几何概括程度加贴图质感，你在Tripo的表单上没有旋钮能调它。但设计图那一层有。把同一段风格描述原样复制到每一条任务里，出来的16张图就是同一套语言，Tripo照着重建，16个模型也就是同一套语言。巴黎那批23件能摆进同一条街不打架，靠的就是这个。这一条在 § 09里有完整写法。

这条流水线长什么样




```

小王子星球-20260910/
├── 输入/                jobs*.jsonl + 22张设计图PNG
├── 产出/                19个从Tripo拿回来的GLB
└── 网页/assets/tripo/进引擎的17个文件, 98 MB

```

22、16、19、17这四个数不一样，先把口径接上，不然你数自己手上的文件会数不对。

输入那22张PNG里，真正送进Tripo的只有16张。另外6张分别是：玫瑰和猴面包树苗各一张备用重做图（原版重建得就对，没送）、老式飞机的设计图（图出了，但这一件始终没送生成）、小王子的一张重编码副本，以及星球表面的两张等距圆柱贴图——那两张是贴到代码生成的球体上去的，本来就不走建模这条路。

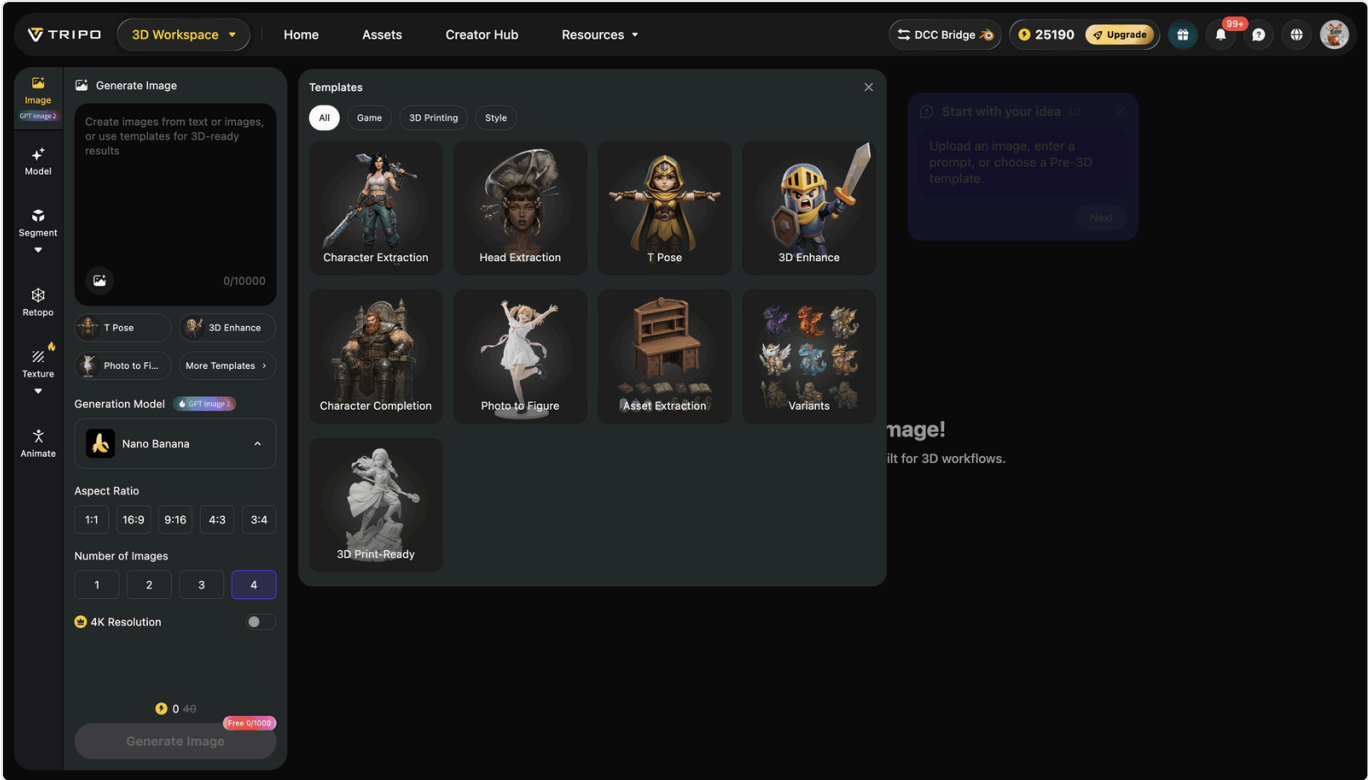
产出那19个GLB = 16件成品 + 小王子绑骨链路上多出来的3个文件。那3个是：带骨架的 `prince-rig.glb`、带 `preset:biped:walk` 行走动画的 `prince-anim.glb`，以及和它同内容的一份副本 `prince-walk.glb`。

最后进引擎的17个 = 15件静态资产 + 绑骨版小王子 + 那个动画文件。没绑骨的原始 `prince.glb` 和那份重复的副本没上场，留在 `产出/` 里。

时间成本比我想的短。小王子那批第一组9张设计图，从21:57跑到22:07，串行，十分钟全部出货（文件时间戳可查，平均一张一分多钟）。Tripo那边单件生成实测各约4分钟。

也就是说，一个下午做16件资产是真的，不是宣传口径。真正吃时间的是决定要哪16件，以及生完之后把它们摆进世界，后面 § 16到 § 19全在讲这个。

这一步Tripo站内就有，不一定非要在站外做



Tripo Studio左栏第一个工具就是Image，角标写着GPT Image 2，2026-09-11截图。右侧Templates面板按Game / 3D Printing / Style分类，里面直接就有T Pose、Asset Extraction、Character Extraction、3D Print-Ready这些冲着3D下游去的模板。左栏还能选生成模型（这张截图上选中的是Nano Banana）、画幅、一次出几张。

我自己两批都是在站外跑的，因为我要的是一个能写进JSONL、一条命令跑16张的批处理入口，而不是在网页上一张一张点。但如果你只做三五件，站内这条路更短：出图和生成3D在同一个工作台里，图出来直接送进Model，中间不落地。

那几个模板的名字我盯着看了一会儿：T Pose、Asset Extraction、3D Print-Ready。给3D用的图是一个独立品类，产品自己也这么认。上一段我论证的那些减法，就是这几个模板替你按下去的开关。§ 09讲的是怎么用文字把同样的开关按得更准、更可复制。

小王子那批：16张图进去，16个模型出来



第一组9张设计图（编号01-09），GPT-image-2出图，1024×1024，统一浅米灰底、略高于平视的四分之三视角。这张是把9张PNG拼成的总览图，原图各自独立。2026-09-10 22:07跑完。



第二组7张道具设计图（编号31-37），对应原作里七颗星球上大人们的東西。同一套风格描述，只换最后一句「画面内容」。2026-09-10 23:13跑完。

这16张图送进Tripo，选v3.1高精度，一件50积分，16件全部一次通过，没有一件重跑。

这句话我不是凭印象说的，它有两道互相独立的证据。**第一道在Tripo那边**：16个任务详情页逐个看过，形体全对，没有一件需要换图重来（下一小节讲怎么用它自动起的名字十秒钟看完一件）。**第二道在引擎这边**：小王子那个demo每次改完我都跑一遍自检，脚本逐件确认GLB有没有真的加载进场景，最后一轮的结论原样写在项目的成果总览里——「**资产到位16/16，零报错**」。

两道证据查的不是同一件事。Tripo那边证明的是「生成对了」，引擎这边证明的是「16件一件不落地进了世界」。这两件事我都裁过，所以现在都查。

钱也对得上：16次生成×50，加两次自动绑骨20，这条线一共840积分，逐笔在附录C里。绑骨那20分之所以扣了两次，是因为那一次第一次没绑成、重试了一遍。**这不是每次都要付的固定开销**——09-12那件A-Pose行人一次就过，两个样本方向正好相反，别照着「每件40分」排预算（\$ 06）。那次重试花在绑骨上，不在16次生成里，上面那句「16件一次通过」说的是生成这一步。**16件资产、一个下午、八百多积分，这是「先出设计图」这条路完整跑通一次的真实价码。**

读Tripo自动起的名字，是最便宜的一道验收

模型生成完，Tripo会给它起一个英文名。关键在于：那个名字不是从你的提示词里抄的，是它照着自己重建出来的形体写的。所以它等于模型自己填了一张验收单交给你。

我要的	Tripo认成了什么
玫瑰	red rose with green stem and leaves, botanical illustration
猴面包树苗	seedling plant with three leaves and pale yellow-green stem
猴面包树成株	gnarled tree with wide trunk and multiple extending branches
狐狸	fox illustration with orange fur, white chest and black legs
蛇	yellow coiled snake with extended tongue
绵羊	fluffy sheep with cream wool, beige ears and brown legs
水井	stone well with wooden frame, rope, bucket and hand-crank winch
老式路灯	ornate street lamp with lantern, gray metal, warm glow
王座	throne-chair-with-red-upholstery-and-crown-finial
镜子	golden-round-vanity-mirror-on-pedestal-stand
酒瓶桌	round-wooden-table-with-green-bottles-on-top
商人书桌	wooden-writing-desk-with-open-map-book-globe-and-inkpot-with-quill

我要的	Tripo认成了什么
地理学家书桌	wooden-table-with-open-book-and-small-wooden-box
箱子	wooden-crate-with-plank-panels-and-nailed-joints
椅子	wooden-chair-with-ladder-back-four-legs-and-square-seat
小王子本人	young boy character with blue coat, red scarf, beige pants and black shoes

看水井那一条：木架、绳子、木桶、手摇绞盘，四样东西一样不少，而我在提示词里写的正是这四样。不用下载GLB、不用开渲染器、不用等模型加载，扫一眼名字就知道它理解成了什么。

16个名字全中。这道验收就发生在任务详情页上，不花钱，也不用等。

注意

这道验收只测「有没有」，不测「像不像」。名字里出现的東西，模型上一定有；反过来不成立：名字写得很确定，形体照样可能是错的。§ 09里梵高那棵柏树被命名成 rocky spire 3d model（岩石尖塔），名字很诚实地告诉了我它理解错了，但如果我要的真是一座尖塔，这个名字看起来完全没问题。

所以正确用法是两段：名字不对，一定有问题，直接重来；名字对了，再去看Tripo自己的预览。

补一条09-12才发现的：这个名字每次都会重写一遍

上面那张表是2026-09-10做小王子那批时记的。两天后我拿巴黎那批设计图重跑了一遍智能网格，才发现一件当时没注意的事：这句英文描述是Tripo每次生成现写的，不是存在某个地方复用的。同一张输入图跑两次，它会给你两句不一样的话。

最干净的证据是凯旋门。同一张 building-06-arc.png，我因为面数参数没设对重跑了一次，两次拿回来的名字是：

跑第几次	Tripo写的描述
第一次	arc de triomphe
第二次（同一张图）	grand arch

一次认出了这是凯旋门，一次只说「一座大拱门」。两次都对，措辞不一样。所以这个名字能用来判断「它理解成了什么」，不能用来当资产的身份标识。

埃菲尔铁塔

巴黎·街景资产

先出图，再建模

1 提示词

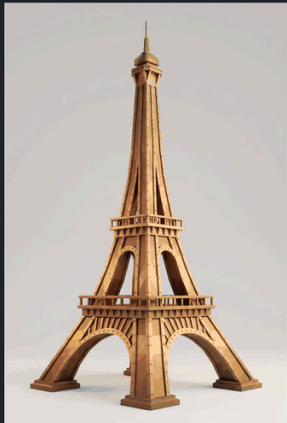
风格：低多边形写实卡通风游戏资产原画
..... 适合image-to-3D重建。

- 单个主体居中构图整塔顶天立地全身入画
- 纯浅灰色背景无阴影无文字水印
- 正面略偏3/4等距透视

画面内容：巴黎埃菲尔铁塔，塔身颜色必须是官方漆色**brun tour Eiffel暖褐铜色**（红棕偏古铜的暖色金属，绝对不是蓝色不是紫色不是灰色），锻铁格构桁架结构网格清晰可辨，四条向外张开的弧形塔腿，底层连接四腿的大装饰拱，第一层与第二层观景平台，收窄的塔尖与顶部天线，塔基四个方形基座。

jobs.jsonl 第 1 行

2 设计图·GPT-image



eiffel-tower-ref.png

3 生成结果·Tripo



eiffel-tower.glb

• 一个物体 • 白底 • 四分之三视角 • 整个东西都在画面里

Tripo·图生 3D

另一件更有意思的：我在提示词里把塔身颜色写死成官方漆色「**brun tour Eiffel暖褐铜色**」，还补了一句「绝对不是蓝色不是紫色不是灰色」。图和模型出来颜色都对了，Tripo给它起的名字是 **wooden eiffel tower**——它把这个暖褐色读成了木头。名字里那个「**wooden**」不是错误，是它对颜色的诚实转述。

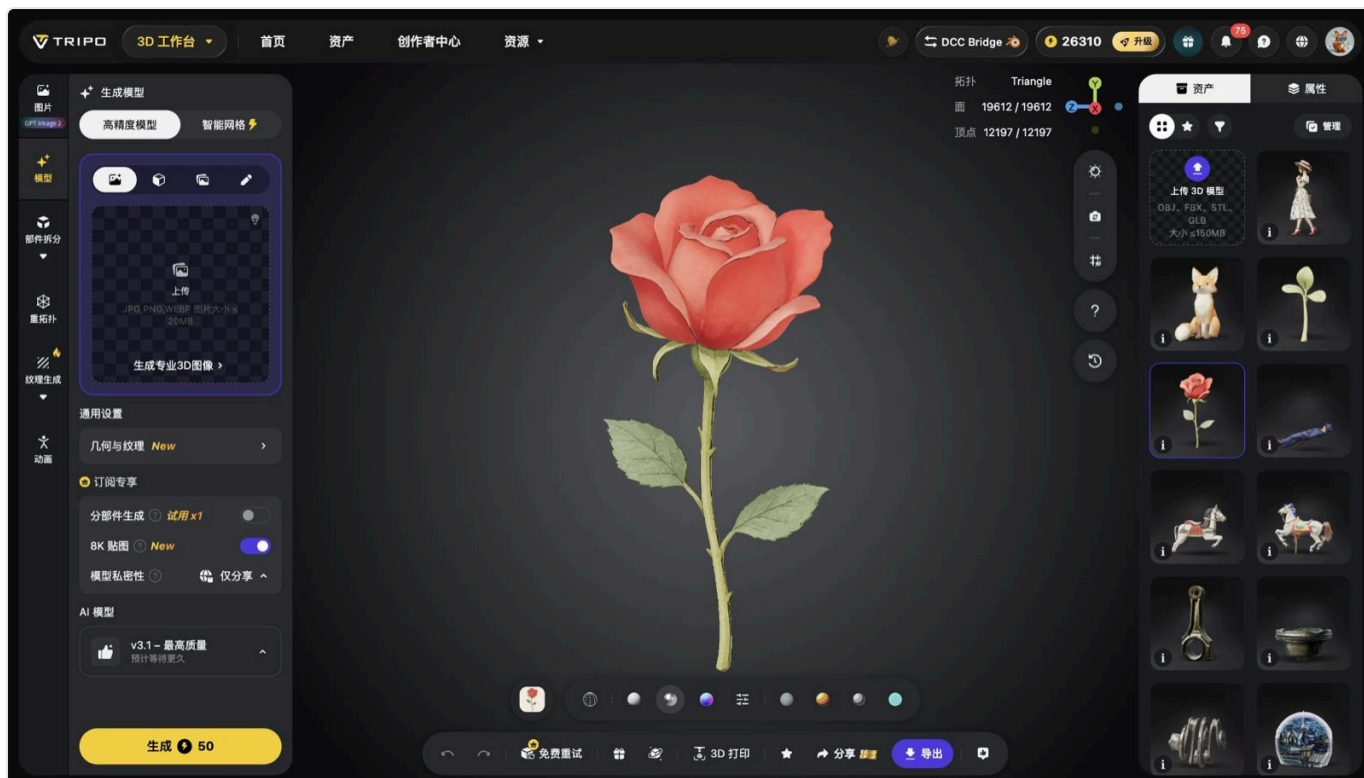
还有一处更容易吃亏的。我那两棵树，提示词写的都是巴黎街头的梧桐，一棵绿冠一棵秋黄。拿回来的名字是：绿冠那件叫 **maple tree**（枫），秋黄那件叫 **oak tree**（橡）。两个名字里都没有「绿」和「秋」，而枫在大多数人脑子里恰恰是「秋天那棵」。三个月后你按名字去认这两个文件，十有八九会装反。

避开的办法就一个，也很便宜：对号入座只认你自己那条提示词和渲图，不认这个名字。我后来给38件重新对号时定的规矩是：文件名由我起（**tree-01-plane-green**、**tree-02-plane-autumn**），Tripo起的那句话只当参考写进表格的备注列。§ 10讲命名的时候会把这条规矩展开。

核心建议

这条验收该怎么用，最后收成三句：名字里出现的东西，模型上一定有；名字里的措辞，下次跑会变；名字不能当文件名，更不能当对号入座的依据。要对号，去读你自己那条提示词。

顺带说，这件事正好从反面说明了「先出设计图」为什么值：你手上那张图是这条流水线里唯一稳定不变的东西。Tripo的措辞每次会飘，模型的面数每次会差一点，只有那张PNG和那行提示词，三个月后打开还是原样。所以我的项目目录里永远是「设计图和 **jobs.jsonl** 跟GLB放在一起」，丢了模型能重生，丢了图就真丢了。



Tripo Studio 3D工作台的玫瑰预览，2026-09-10。右上角写着面19612 / 顶点12197，「面」指三角面，是模型的复杂度单位。左栏能看到这次用的是「高精度模型」+ AI模型「v3.1 - 最高质量」，右下角生成按钮标价50积分。右侧资产栏里排着同一批的狐狸、树苗、旋转木马。

巴黎那批：23件走同一套流程，留了设计图的19件全部一次通过



巴黎转角公寓设计图（building-01-corner.png），1024×1024。同一段风格前缀写死了「奥斯曼巴黎复古配色」「纯浅灰色背景无阴影无文字水印」「3/4等距透视单个主体居中构图」，9栋建筑逐条复用。

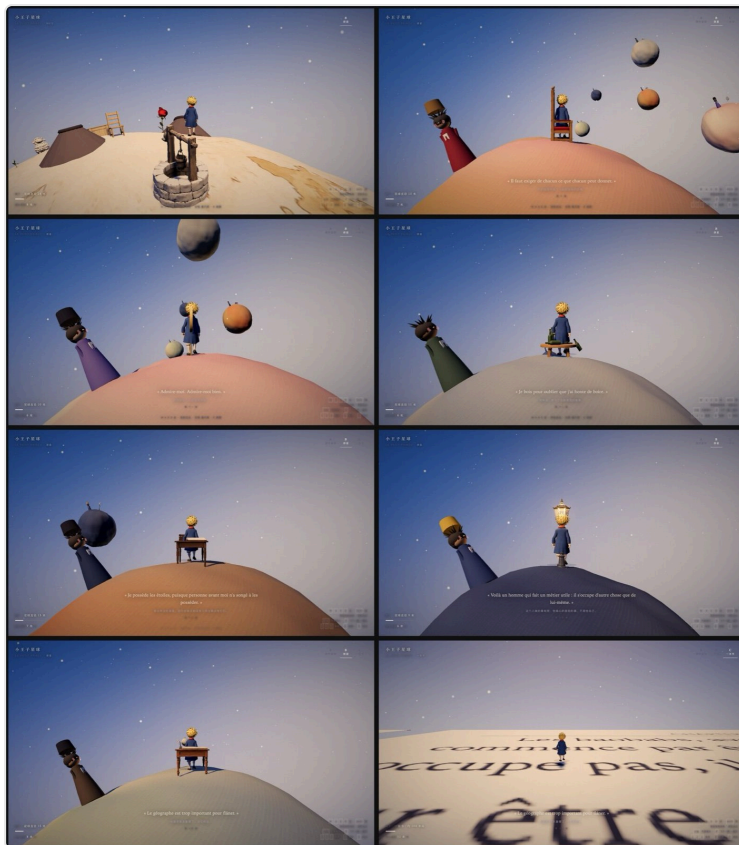


巴黎复古出租车设计图 (car-01-taxi.png)。提示词里写的是「雷诺4CV造型」和「与经典雪铁龙2CV同一美术风格」，用真实车型名锚定，比堆「复古」「可爱」这类形容词准得多。

开发笔记里我当时写的原话是：

23个资产（建筑、人物、载具、道具）用同一套Codex出图 → Tripo image-to-3D的流程跑下来，材质质感、比例、可辨识度全部在线，没有一个明显失败要重跑（19/19用于最终游戏的资产一次通过）。

这里要把话说准，免得你以为只有19件进了游戏。23是最终发布包里点得到的Tripo模型总数，\$ 16和\$ 19用的都是这个数。19是其中留了设计图和提示词、你照着能复现的那一批，附录A的A组B组给的就是这19件。差的4件（雪铁龙2CV、埃菲尔铁塔、奥斯曼公寓、街角咖啡馆）是我直接在网页里点出来的，当时没留输入图。所以那句「19/19一次通过」说的是有据可查的那19件。



小王子星球道具自检拼图，2026-09-11。脚本逐件把摄像机摆到固定位置出图，用来确认「16件是不是真的都进世界了、比例对不对」。七颗星球上的王座、镜子、酒瓶桌、两张书桌、路灯，每件都能在画面里认出来。

什么时候可以跳过设计图

输入本来就是一张真图的时候。

名画那两件（星月夜里的柏树、千里江山图的一段山）是从3840 px原作上手工抠下来的；发动机的活塞和连杆是零件照片裁的。这些都没有设计图这一步，直接送进Tripo。判据很简单：你要的东西世界上已经有一张正面清楚的图，就别再让AI重画一遍。重画只会让它离你想要的更远，而且多花一次生成的时间。

反过来，只要这个东西是你脑子里、现实中没有现成正面图的，设计图这一步就省不掉。小王子星球上那16件全属于后者，所以16件全都先画了一遍。

下一节把这条流水线上最需要手艺的那一半拆开讲：那张设计图的提示词到底怎么写。

§09 提示词技巧示例（含图）

Prompt Recipes for Image-to-3D

十八条从真实项目里逐字抄出来的写法，每条尽量配「提示词原文+设计图+ Tripo结果」三联。素材是我这两周真跑过的95条提示词，一条没编。包括两个翻车的：柏树被理解成岩石尖塔，活塞被理解成一枚硬币。它们留在这儿是因为两个坑都有明确的避法。

先说一句总的判断：**给Tripo用的图写提示词，和给人看的图写提示词，是两件事。**

给人看的图要好看，光影、景深、氛围都是加分项。给Tripo看的图要好读——它要从一张2D图里猜出背面长什么样，任何会干扰它读形体的东西都是减分项：投影、渐变、高光、背景、被裁掉的一半、贴在身上分不开的手臂。

所以下面这十八条，大半是在做减法。

核心建议

这一节的例子全部来自项目里那10个 `jobs*.jsonl` 文件，合计95条提示词。其中8个文件、60条随包发，在资源包的 设计图与提示词/ 里，可以直接打开那些文件改，不用从这一页往外抄；剩下的巴黎第三批那32条和P2.0重生成那3条没赶上09-11封版（理由见 § 00「包里没有什么」）。本节凡是引到第三批的地方，提示词原句都整段抄在正文里了；P2.0那3条本节没有引用。

01 · 四条配方：一个物体、白底、四分之三视角、整个东西都在画面里

这四条是所有例子的公共前缀，一条都不能少。如果这一节你只抄一样东西，抄它。

单个物体、完整入画、居中放在纯白背景上，四分之三视角，只有极淡的接触阴影、无其他物体、无文字、无人物。

这是游乐园那批8条提示词共用的那半句，一字不差。同一套要求在小王子那批写成「纯浅米灰背景，无地面投影、无文字、无水印，单个主体居中完整入画，略高于平视的3/4视角」，在巴黎那批写成「纯浅灰色背景无阴影无文字水印，3/4等距透视单个主体居中构图」。三批的说法不同，四项要求是同一套。

鬼屋

游乐园 · 游乐设施

先出图，再建模

1 提示词

皮克斯动画电影的场景设定图：

- 独栋建筑
- 居中放在纯白背景上
- 四分之三视角
- 完整入画

一座游乐园鬼屋小楼，紫灰色木板墙、墨绿尖顶、歪斜的窗框和歪掉的烟囱、门口有南瓜灯。

jobs.jsonl 第 5 行

2 设计图 · GPT-image



05-鬼屋.png

3 生成结果 · Tripo



haunted-house.glb

● 一个物体

● 白底

● 四分之三视角

● 整个东西都在画面里

Tripo · 图生 3D

四条配方全中的一件：`05-鬼屋.png` → `haunted-house.glb`。注意它的提示词把第一条改写成了「独栋建筑、完整入画」。主体是一栋楼的时候，「一个物体」这句话要换个说法它才听得懂，这一点第10条会展开。

四条各自在挡一件具体的事，挡的都不是同一件：

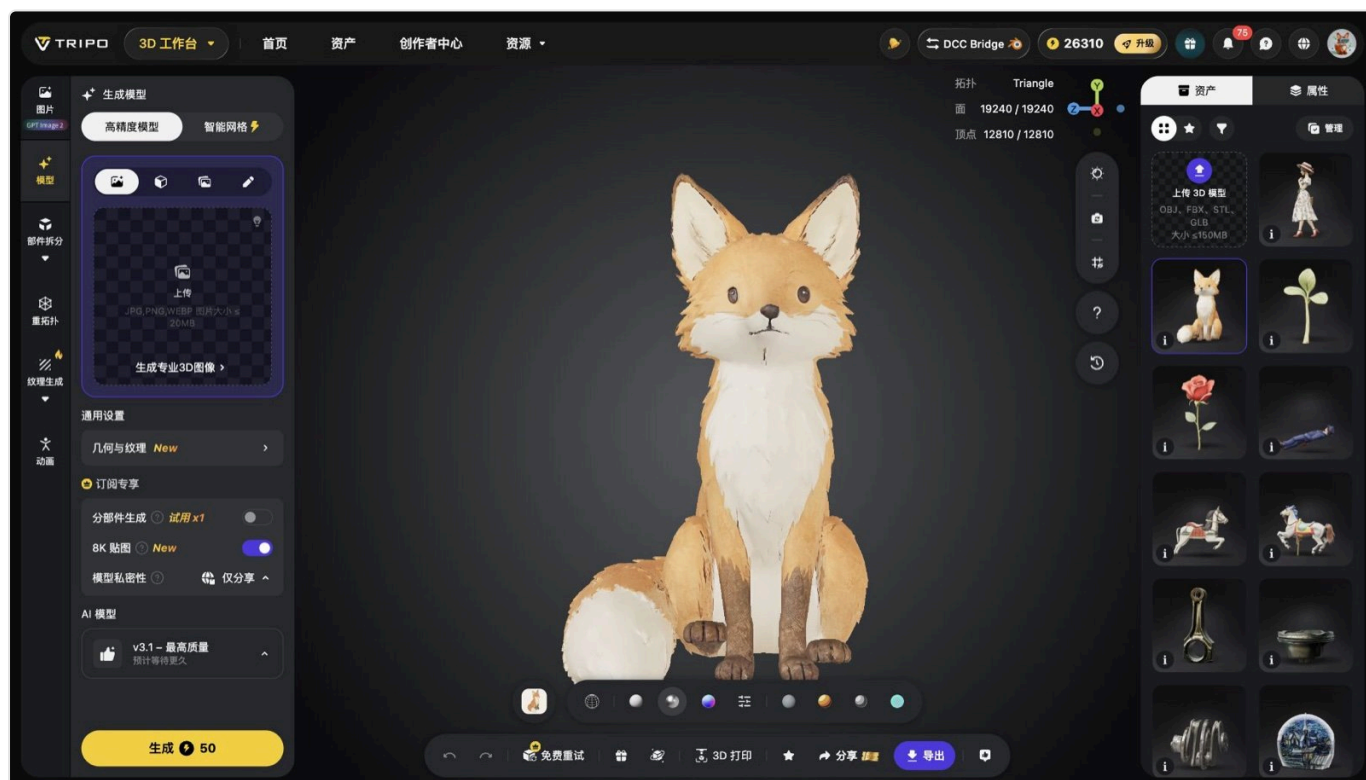
配方	不写它会怎样
一个物体	画面里出现第二件东西，你不知道它会把哪个当主体重建
白底、无投影	投影会被当成几何读进去，模型脚下多一块莫名其妙的板
四分之三视角	正视图只给一个面的信息，重建出来的背面会塌
整个东西都在画面里	被裁掉的部分它不会补，会按看到的比例老实重建（第06条那枚硬币就是这么来的）

下面这条是小王子那批的完整写法，可以看到公共前缀和「画面内容」是怎么拼的：

风格：安托万·德·圣埃克苏佩里《小王子》插图的语言—深色钢笔线条勾出干净轮廓，内部平涂低饱和水彩色块，暖米白纸面质感，形体概括、块面分明，没有复杂纹理、没有高光渐变，适合 image-to-3D 重建。纯浅米灰背景，无地面投影、无文字、无水印，单个主体居中完整入画，略高于平视的 3/4 视角。画面内容：一只安静坐着的狐狸，橘棕色皮毛，白色的下巴和肚皮，两只尖耳朵，深色的四只爪子，一条粗大的尾巴盘在身体侧面。



设计图 04-狐狸.png，GPT-image-2，1024×1024，2026-09-10 22:01。



Tripo结果，v3.1高精度，面19240/顶点12810。Tripo给它起的名字是 fox illustration with orange fur, white chest and black legs，橘毛、白胸、黑腿，三样都在。

我在小王子这批用的是浅米灰不是纯白，因为纯白容易和物体上的高光区混在一起。但要点是「纯色、无渐变、和主体不同色」，不是某个具体颜色。拿不准就照第一段那句原文写纯白，那是最稳的。

02 · 风格段落原样复制到每一行，一个字都别改

上面那段提示词里，从「风格:」到「3/4视角。」是完全固定的，16条任务一字不差，只有「画面内容:」后面那句在变。巴黎那批也一样：

风格：低多边形写实卡通风游戏资产原画，奥斯曼巴黎复古配色（奶油色石材/藏青灰蓝屋顶/黑色锻铁/黄铜细节），暖调柔和摄影棚布光，纯浅灰色背景无阴影无文字水印，3/4等距透视单个主体居中构图，细节清晰适合image-to-3D重建。
画面内容：一间巴黎街角面包店建筑门面，红色帆布雨棚，金色手写体招牌字体位置留白，橱窗摆放法棍面包造型剪影，深木色门框。



设计图 building-03-boulangerie.png。这段风格前缀在 trip-source/jobs.jsonl 的19行里出现了19次，除了角色和载具那几条把「资产原画」换成「角色原画/载具原画」，其余全部逐字相同。

我在巴黎的开发笔记里把这一条写成了可复用的结论：

统一浅灰背景+ 3/4等距视角+风格关键词一致，是保证一批资产美术风格统一的关键，缺了任何一条，批量生成出来的东西会互相打架。

打架是很具体的一件事：一栋楼是柔和布光的，隔壁那栋是硬光带投影的，摆在同一条街上，你说不清哪里不对，但就是假。风格统一不是审美追求，是这批资产能不能摆在一起的前提。

要换词头的只有一处，就是「资产原画/角色原画/载具原画/植被资产原画」这几个字。它们决定了出图模型把这次任务归到哪一类，换掉之后人物的比例和载具的透视会明显更对。除了这几个字，风格段落里剩下的每一个逗号都别动。

03 · 用真实参考名替代形容词，这是整节最划算的一招

如果说第02条决定一批资产协不协调，这一条决定单件像不像。

出租车

巴黎 · 街景资产

先出图，再建模

1 提示词

风格：低多边形卡通游戏载具原画
..... 适合image-to-3D重建。

· 单体居中构图
· 纯浅灰色背景无阴影无文字水印
· 3/4等距透视

画面内容：一辆巴黎复古出租车，
雷诺4CV造型，黑色车身搭配奶油
黄车顶，车顶装有TAXI字样灯箱位
置留白。

jobs.jsonl 第 15 行

2 设计图 · GPT-image


car-01-taxi.png

3 生成结果 · Tripo


car-01-taxi.glb

◆ 一个物体 ◆ 白底 ◆ 四分之三视角 ◆ 整个东西都在画面里

Tripo · 图生 3D

出租车这一件：提示词→ car-01-taxi.png → car-01-taxi.glb。这张图后来成了全书跑得最多的一张输入图，两种几何模式的对照实验（§ 04）用的也是它。

它的提示词里有两个真实车型名：

风格：低多边形卡通游戏载具原画，与经典雪铁龙2CV同一美术风格，暖调柔和摄影棚布光，纯浅灰色背景无阴影无文字水印，3/4等距透视单体居中构图，细节清晰适合image-to-3D重建。画面内容：一辆巴黎复古出租车，雷诺4CV造型，黑色车身搭配奶油黄车顶，车顶装有TAXI字样灯箱位置留白。

「雷诺4CV造型」定的是形体，「与经典雪铁龙2CV同一美术风格」定的是美术语言。这两个词组合起来，比你写十个形容词都准。因为形容词要经过模型的一次翻译，专名不用，它直接指向一个世界上存在的、有确定外形的东西。

「复古」这个词模型能给你1920年代，也能给你1970年代；「雷诺4CV」只有一个答案。我在巴黎那批里到处用这一招：报刊亭写「Kiosque à journaux」，地铁口写「Guimard风格」，喷泉写「Fontaine Wallace」，广告柱写「Colonne Morris」，咖啡椅写「Maison Drucker」，绑骨姿势写「参考Mixamo角色T-Pose」。

报刊亭

巴黎·街景资产

先出图，再建模

1 提示词

风格：低多边形写实卡通风游戏资产原画 适合image-to-3D重建。

- 单个主体居中构图
- 纯浅灰色背景无阴影无文字水印
- 3/4等距透视

画面内容：一座巴黎经典报刊亭（Kiosque à journaux），深绿色铸铁圆顶结构，八边形亭身，顶部小尖塔装饰，复古报纸杂志陈列窗。

jobs.jsonl 第 5 行

2 设计图·GPT-image



building-05-kiosque.png

3 生成结果·Tripo



building-05-kiosque.glb

● 一个物体 ● 白底 ● 四分之三视角 ● 整个东西都在画面里

Tripo·图生3D

报刊亭这件把专名和几何描述叠在一起用：先写「巴黎经典报刊亭（Kiosque à journaux）」锁住这是哪一种东西，再补「深绿色铸铁圆顶结构，八边形亭身，顶部小尖塔装饰」把它的几何说死。前一句负责像，后一句负责能建。

用法上有个分寸。专名负责「是什么」，几何描述负责「怎么建」，两个都要写。只写「Kiosque à journaux」，出图模型可能给你一个正面平视的写实照片；只写「八边形深绿铸铁亭子」，出来的东西谁也认不出是巴黎的。

核心建议

挑专名的标准：这个名字在图片搜索里的头二十张结果，是不是长得都差不多。是，就能用；不是，说明这个词在模型脑子里也是发散的，换一个。

注意

专名是给出图模型看的，不是给最终资产挂的。上面这些名字（雪铁龙2CV、雷诺4CV、Maison Drucker、Mixamo）在提示词里只干一件事：把「长什么样」说死，让模型少猜一次。它们不该出现在你的文件名、材质名和贴图上的文字里，也不该被当成这件资产的出身说明。所以这一条和§00那份剔除标准不打架：那份标准剔的是模型文件本身（包里就没收直接叫2CV的那件车，逐条理由在附录A末尾），管的不是你写提示词时拿谁当尺子。真要把成果拿去商用，商标这一层请自己核一遍——本节只负责教你怎么让模型听懂你要的形状。

04 · 一个准确的术语，能打包掉一整段解释

上一条是拿专名指物，这一条是拿术语指做法。

小王子那颗星球的表面不是建模出来的，是一张糊上去的贴图。这类需求最难说清楚的是「展开方式」，而它有一个精确的词：

.....画面内容：一张星球表面的地图，采用等距圆柱投影（赤道横贯正中，上下两极的区域自然收拢淡出），左右两个边缘可以无缝拼接成一圈。

「等距圆柱投影」是世界地图最常见的那种展开方式。写这六个字，比写十句「要能包住一个球、左右要接得上、两极要收进去」都管用，而且一次到位。

同一类的还有：「T-Pose」「低多边形」「PBR」「孟莎屋顶」「掌状复叶」「等距透视」。形容词打不了这个包，术语可以。写提示词的时候我会专门停一下问自己：我正在描述的这件事，有没有一个行业里的固定叫法？有就用它，那一句能省掉三行。

05 · 角色要绑骨，参考图得按绑骨的规矩来

绑骨（给模型装上一副可以驱动的骨架，之后才能播走路、挥手这类动画）对参考图的要求比静态资产严得多。我头两次都没成，第三次Tripo自己把答案弹在了页面上。

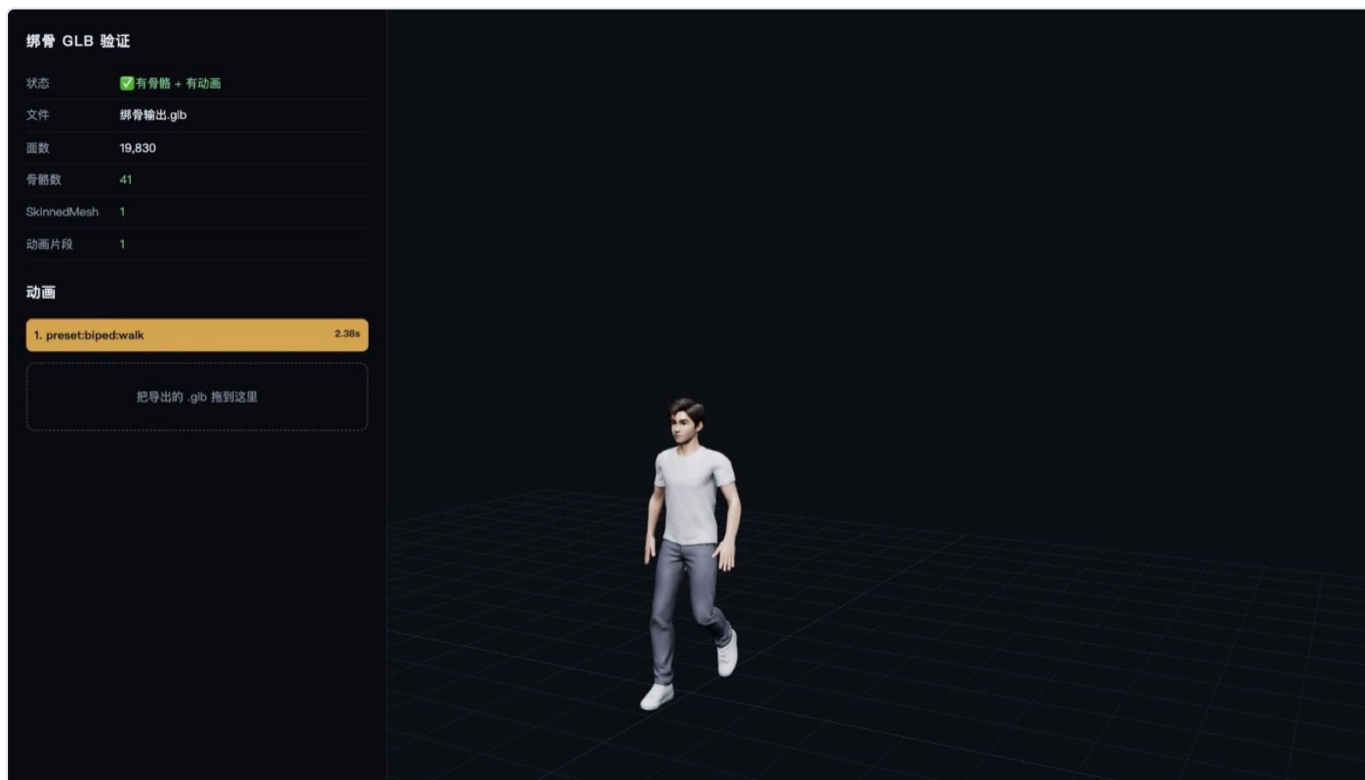


Tripo Studio的官方提示，2026-09-10。这个模型（面19478 / 顶点12260）是一位走路姿态的碎花裙女士，站得挺直但一只手搭在胸前，绑骨没成。顶部横幅写着：「为了获得更好的绑定效果，请上传T-Pose图像并重新生成模型。」

T-Pose就是双臂水平张开站成一个「大」字的姿势，是骨骼动画行业的通用起手式。我前两次没成的原因，不是姿态不够正，是根本就不是T-Pose：第一次用的是走路中被拍到的前倾侧身行人，第二次用的是站直但手臂弯着的女士。



重新生成的T-Pose参考图，2026-09-10。写提示词时我用Mixamo当锚点，它是干净可绑骨角色这件事最标准的参考名。



这张T-Pose图走完「Tripo图生3D 50 → 自动绑定20 → 动画库选行走 → 导出」，在Three.js里的验证结果：面数19,830、骨骼41、SkinnedMesh 1、动画片段1个 `preset:biped:walk`，时长2.38秒。截图里角色正在迈腿，不是滑行。

小王子本人是主角，要走路，所以他那一张的提示词是全批里最长最啰嗦的一条，啰嗦的地方全在四肢的分离度上：

.....画面里没有复杂纹理、没有高光渐变、没有细长垂挂物，适合 image-to-3D 重建与自动骨骼绑定。纯浅米灰背景，无地面投影、无文字、无水印，单个主体居中完整入画，正视角，全身都要在画面里。画面内容：一个全身站立的小男孩，头顶是蓬松的金色短发，脸上只有两个小小的墨点当眼睛；穿一件及膝的深蓝色长外套，外套下摆不超过膝盖，里面是浅米色衬衫；脖子周围围着一圈红色短围巾，围巾就搭在脖子上、不要长出飘带；两条手臂自然下垂但向外张开约三十度，手臂与躯干之间留出清楚的空隙，不贴住身体；两条腿分开站立、间距明显，两腿之间和腿与外套之间都能看出分开；脚上是两只小小的深色鞋子。整体比例略偏卡通，头稍大；关键是四肢轮廓要清楚可分离。



小王子这一件的三联。注意中间那张图的三件事：正视角（不是四分之三）、手臂离开躯干、两腿之间有缝。这三条都是为了让骨架算法能把四肢切开。

为什么要写「围巾就搭在脖子上、不要长出飘带」？因为飘带是细长垂挂物，绑骨算法会犹豫它属于身体还是属于手臂。绑骨用的参考图，画的不是好看的角色，是一副容易被切开的轮廓。

还有一件事值得单说，因为它把我自己原来的判断修正了一半。真正承重的不是「严格水平的T-Pose」，是「四肢和躯干分得开」。2026-09-12我又做了一个会走路的行人，那张参考图上手臂是向下斜着张开三四十度的A-Pose，不是水平的T，自动绑定19秒，第一次就成。那件的细节在 § 06。

所以参考图的硬要求，我最后收成四条：全身入画、正视角、手臂向外张开与躯干留出清楚空隙、两腿分开站稳。手臂是水平还是下斜三十度，没那么要紧。

补一句：Tripo站内的Image工具里直接有一个叫T Pose的模板（那张面板截图在 § 08）。我是先踩了两次坑才反推出这条规矩，后来才看见它就摆在那儿。你要是只做一两个角色，从那个模板起步比自己写这段提示词省事得多；要批量做、要每次都一样，还是得把上面这段写死。

06 · 比例不写进提示词，就等于交给了截图

这是我踩得最疼、也最容易被忽略的一条。



送进Tripo的活塞参考图。这是从一张真实零件照片上裁出来的，白底，但裙部（下半截）被裁掉了。

结果Tripo老老实实按它看到的比例重建：一个扁扁的圆盘。做四冲程发动机demo时我在笔记里写的是：直接用会像一枚硬币。

这一件我想说清楚是谁的问题：**不是重建不准，恰恰是太准了**。我给它的图里那个活塞就是扁的，它照着做了。配方第四条「整个东西都在画面里」，说的就是这件事：**被裁掉的部分，它不会替你脑补**。

怎么避开，两条路，我两条都用了：

1 事前：换一张没被裁的图，或者把比例写进提示词

如果这件东西是你自己出的设计图，就在提示词里写死比例意图。巴黎那批树的提示词里专门有一句「3/4视角单棵树完整居中构图，树干与树冠比例正确」，就是冲着这个去的。

2 事后：在引擎里把它拉回来

已经生成了、又不想再花一次钱，就在代码里单轴拉伸。活塞那件我在引擎里**纵向拉了1.75倍**，而且只动Y方向。这招对回转体（活塞、柱子、瓶子）有效，对有明确轮廓的东西（车、人、房子）会拉变形，别用。



拉伸修正后的活塞（和连杆一起）在发动机demo里的样子，2026-09-10。右侧零件表标着每一件的来源：活塞、连杆是Tripo，曲轴、缸筒、气门、火花塞是代码。

教训不是别裁图，是你没在提示词里说的比例，模型就从图里量。所以巴黎那批树的提示词长这样：

风格：低多边形写实卡通风游戏植被资产原画，奥斯曼巴黎复古配色，暖调柔和摄影棚布光，纯浅灰色背景无阴影无文字水印，3/4视角单棵树完整居中构图，树干与树冠比例正确，细节清晰适合image-to-3D重建。画面内容：一棵巴黎街头梧桐树，高大笔挺的浅灰色斑驳树干，茂密宽大的掌状绿色树冠，树冠饱满有层次。



设计图 tree-01-plane-green.png。「单棵树完整居中构图」加「树干与树冠比例正确」，这两句都是冲着比例去的。行道树最容易出的错是树冠占满画面、树干只剩一小截，摆进街景里就变成一朵蘑菇。

07 · 数量和姿态要数出来，别写「一群」

.....画面内容：一小群巴黎街头鸽子，五只灰蓝色鸽子聚在一处，其中两只展翅姿态、三只站立啄食，造型低多边形简化但可辨识。



设计图 prop-06-pigeon-flock.png。五只、两只展翅、三只啄食，数字写死了，出来就是这个数。

「一群鸽子」这种写法，模型可能给你三只，也可能给你三十只，而三十只的重建结果是一团糊在一起的灰块。多物体本来就是image-to-3D的难区（第01条那句「单个主体居中完整入画」就是为这个写的），真要做一组，就把数量和姿态数清楚，让它保持稀疏、每只都能看出轮廓。

08 · 真要两个东西同框，就让它们「紧凑成组」

第01条说「一个物体」，但现实里总有那么几件东西拆开就没意义了：人和他骑的车、桌子和配套的椅子、牵狗的人和那只狗。

骑车的女孩

巴黎·街景资产

先出图，再建模

1 提示词

风格：低多边形卡通游戏角色原画
..... 适合image-to-3D重建。

- 单个角色居中构图
- 纯浅灰色背景无阴影无文字水印
- 3/4视角

画面内容：一位骑着巴黎经典城市自行车的角色，棕色复古自行车配车篮，角色穿休闲装。

jobs.jsonl 第 12 行

2 设计图·GPT-image



pedestrian-03-cyclist.png

3 生成结果·Tripo



pedestrian-03-cyclist.glb

● 一个物体
 ● 白底
 ● 四分之三视角
 ● 整个东西都在画面里

Tripo·图生 3D

骑车的女孩：人和车是一件资产，一起生成、一起进引擎。提示词写的是「一位骑着巴黎经典城市自行车的角色，棕色复古自行车配车篮，角色穿休闲装」。没写「一个人和一辆车」，写的是「一位骑车的角色」。一个主语，两样东西长在一起。

句式上的差别很小，效果差别很大。「A和B」会让它构图成两件东西，「骑着B的A」只有一个主体。巴黎那批的露天桌椅组用的是同一招，提示词末尾还补了一句兜底：

.....画面内容：一组巴黎咖啡馆露天桌椅，一张圆形白色大理石桌面配黄铜包边与铸铁桌腿，两把 Maison Drucker 红白藤编扶手椅一左一右，桌旁立着一台圆柱形不锈钢燃气取暖伞，顶部有反射罩，三件物体紧凑成组不分散。

「三件物体紧凑成组不分散」这一句是我后来加的。它把「多主体」重新定义成了「一个组」，重建出来是一坨连在一起的东西，而不是三件互相飘着的碎片。

什么时候可以破第01条，判据只有一个：这两样东西在你的游戏里会不会分开动。会分开动（车轮要转、门要开）就必须拆，那是 § 05 拆件那一节的事；永远黏在一起（骑车的人、成套的桌椅）就合成一件生成，省一次钱也省一次装配。

09 · 做零件，不是做整台机器

这条是我做游乐园那批时想明白的，后来成了我决定「哪些东西给Tripo做」的主要判据之一。

摩天轮吊舱

游乐园 · 游乐设施

先出图，再建模

1 提示词

皮克斯动画电影的道具设定图：

- 单个物体
- 居中放在纯白背景上
- 四分之三视角
- 完整入画

一个复古摩天轮的吊舱，红白双色车身、球形玻璃窗、黄铜色金属吊架。

jobs.jsonl 第 2 行

2 设计图 · GPT-image



02-摩天轮吊舱.png

3 生成结果 · Tripo



ferris-cabin.glb

• 一个物体

• 白底

• 四分之三视角

• 整个东西都在画面里

Tripo · 图生 3D

整座摩天轮我一件都没让Tripo做，只做了这一个吊舱：「一个复古摩天轮的吊舱，红白双色车身、球形玻璃窗、黄铜色金属吊架。」轮圈、辐条、转轴、旋转角度，全部是代码画的、代码转的。

为什么这么分：圆的、重复的、要动的那部分，代码算得又准又便宜；有造型、有材质、观众会凑近看的那部分，生成模型做得比代码好太多。一座摩天轮的轮圈是一个正多边形加辐条，写代码十分钟，精确到小数点；让AI生成一个几十米高的轮圈，它给你的每根辐条粗细都不一样，还没法转。

游乐园那6件全是按这个思路拆的：旋转木马（做一整座，因为它的顶棚雕花才是看点）、摩天轮吊舱、飞椅的一把吊椅、旋转茶杯的一只杯、鬼屋、海盗船船体。三件是「机器上取下来的那个零件」，不是机器本身。

同一条判据，我在做巴黎那座城的时候是这么用的：**开车会开到跟前细看的东西，用Tripo做；只会远远瞥一眼的，代码画就够了。**这句话在 § 15、§ 14和 § 18各印证一次，是全书最短也最实用的一条工程判据。

10 · 建筑别写「一个物体」，写「独栋建筑」

主体是一栋楼的时候，「单个物体」这四个字会让出图模型犯难：楼本来就是由很多东西组成的。所以游乐园那批的鬼屋和售票亭，第一条被改写了：

皮克斯动画电影的场景设定图：一座游乐园鬼屋小楼，紫灰色木板墙、墨绿尖顶、歪斜的窗框和歪掉的烟囱、门口有南瓜灯。独栋建筑、完整入画、居中放在纯白背景上，四分之三视角，只有极淡的接触阴影、无其他物体、无文字、无人物。均匀柔和的摄影棚布光，颜色鲜明干净。

「独栋建筑」和「单个建筑」这两种写法我都用过，出来都对。关键是把量词换成主体本身的量词：一栋楼、一棵树、一位角色、一辆车、一组桌椅。巴黎那批的树写的是「单棵树完整居中构图」，人物写的是「单个角色居中构图」，都是同一个动作。

136

还有一句是专门给建筑和地标加的，在巴黎第三批那32条里出现了25次（剩下7条是人物，它们换成了「全身从头顶到鞋底完整入画不裁切」，意思一样）：

.....3/4 视角，单个主体居中，完整入画不裁切不被边框切断，细节锐利，适合 image-to-3D 重建。

「不裁切不被边框切断」是「完整入画」的加强版。建筑和铁塔这类又高又瘦的东西，出图模型特别爱把顶部切掉一点点，它觉得那样构图好看。一句「不被边框切断」，比把画幅改成竖版还管用。（铁塔那件我两样都做了：画幅1024×1536，提示词写「整塔顶天立地全身入画」。）

11 · 品牌字样让它留白，别让它写字

巴黎那批里有三处需要文字的地方，我全部写成了「位置留白」：面包店的「金色手写体招牌字体位置留白」、地铁口的「黄色Metropolitain字样招牌位置留白」、出租车的「车顶装有TAXI字样灯箱位置留白」（那条提示词在第03条）。

模型其实写得出那几个字，问题在后面：写上去的字会被烘进贴图，之后改不掉。留白留出来的是一块干净的牌子，进引擎之后想贴什么贴什么，还能做多语言。这一条对做游戏比对做单件模型重要得多。

12 · 「无文字」是倾向不是保证，重要的位置得正面写

上一条说的是主动留白，这一条说的是被动禁止不一定拦得住。

街角咖啡馆

巴黎 · 街景资产

先出图，再建模

1 提示词

风格：低多边形写实卡通风游戏资产原画 适合image-to-3D重建。

- 单个主体居中构图
- 纯浅灰色背景无阴影无文字水印
- 3/4等距透视

画面内容：一栋巴黎街角小咖啡馆建筑，底层深红色帆布雨棚与户外藤编圆桌椅，奶油色石材上层立面，黑色锻铁阳台栏杆，转角切角玻璃门入口，顶部灰蓝色锌板屋顶。

jobs.jsonl 第 2 行

2 设计图 · GPT-image

prop-cafe-shop-ref.png

3 生成结果 · Tripo

prop-cafe-shop.glb

• 一个物体
• 白底
• 四分之三视角
• 整个东西都在画面里

Tripo · 图生 3D

街角咖啡馆这一件，提示词里明明白白写着「纯浅灰色背景无阴影无文字水印」，出图模型还是在门楣上写了一行 LE CHAT NOIR CAFE。它对重建没影响（形体完全正确），但那行字被烘进了贴图，右边模型上能看到它变成了一串糊掉的字形。

这件事的性质要说准：「无文字水印」这类否定句约束的是背景和水印，不是画面主体自己的招牌。模型的理解没错，是我那句话没覆盖到那个位置。它也不影响用：这件资产我照常装进了城里，那行糊字在街对面根本看

不出来。

所以避法是正面写，不是把否定句写得更重：

核心建议

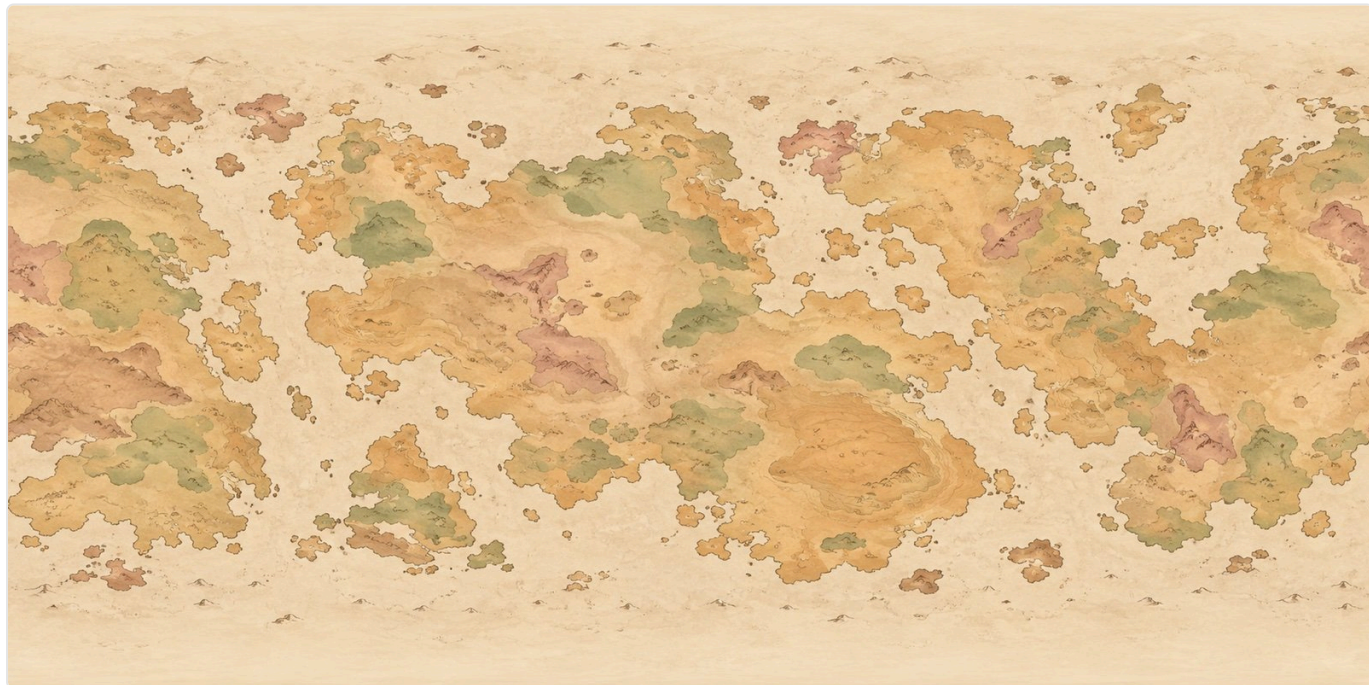
不要写：「无文字、无招牌字、绝对不要任何文字」（写三遍也拦不住主体上的招牌）

要写：「招牌位置留白」「门楣上不写任何文字」，**指名道姓说清楚是哪个位置**。巴黎第三批那32条我就是这么改的：凡是主体上可能长出字的，都单独加一句：红磨坊是「门面上不写任何文字与招牌字」，公交车是「车身上不写任何文字、编号、路线牌与品牌标识」，游船是「船体上不写任何文字与标识」，候车亭是「面板上不写任何文字与图案」，警察是「制服上不出现任何文字、徽章图案与臂章图案」。32条里有12条挂了这种指名道姓的句子，正好就是那12件身上会有字的东西。

13·贴图类资产要告诉它「这是贴图，不是画」

星球表面不是一个模型，是一张要糊到球上的图。这类需求得把用途直接写进提示词，而且要把「画」的本能全部禁掉：

.....画面内容：一张星球表面的地图，采用等距圆柱投影（赤道横贯正中，上下两极的区域自然收拢淡出），左右两个边缘可以无缝拼接成一圈。地表由暖赭石与沙色的缓坡丘陵构成，点缀几片鼠尾草绿的小草坡和两 three 处淡玫瑰色的裸岩，偶尔有几道极细的钢笔线条暗示等高线。这是一张贴到球面上的贴图，不是一张画：整体必须平整、光照均匀、没有强光、没有投影、没有明暗渐变，颜色过渡柔和。画面里不要有天空、云、水、人物、建筑、文字、水印、边框、坐标网格。



设计图 20-星球表面-等距圆柱.png，2048×1024（贴图要2:1才能正好包住一个球）。



同一张贴到球上的效果，2026-09-11。明暗是引擎里的真实光源打出来的，所以贴图本身必须是平的；贴图里要是自带了阴影，就会和真实光照打架，出现两套方向不同的影子。

「这是一张贴到球面上的贴图，不是一张画」这半句，是整条提示词里最值钱的。**把下游用途直接写出来，模型会自己收敛掉一堆你还没想到要禁的东西：戏剧光、暗角、笔触的立体感，全都跟着没了。这个招数在第16条还会再出现一次。**

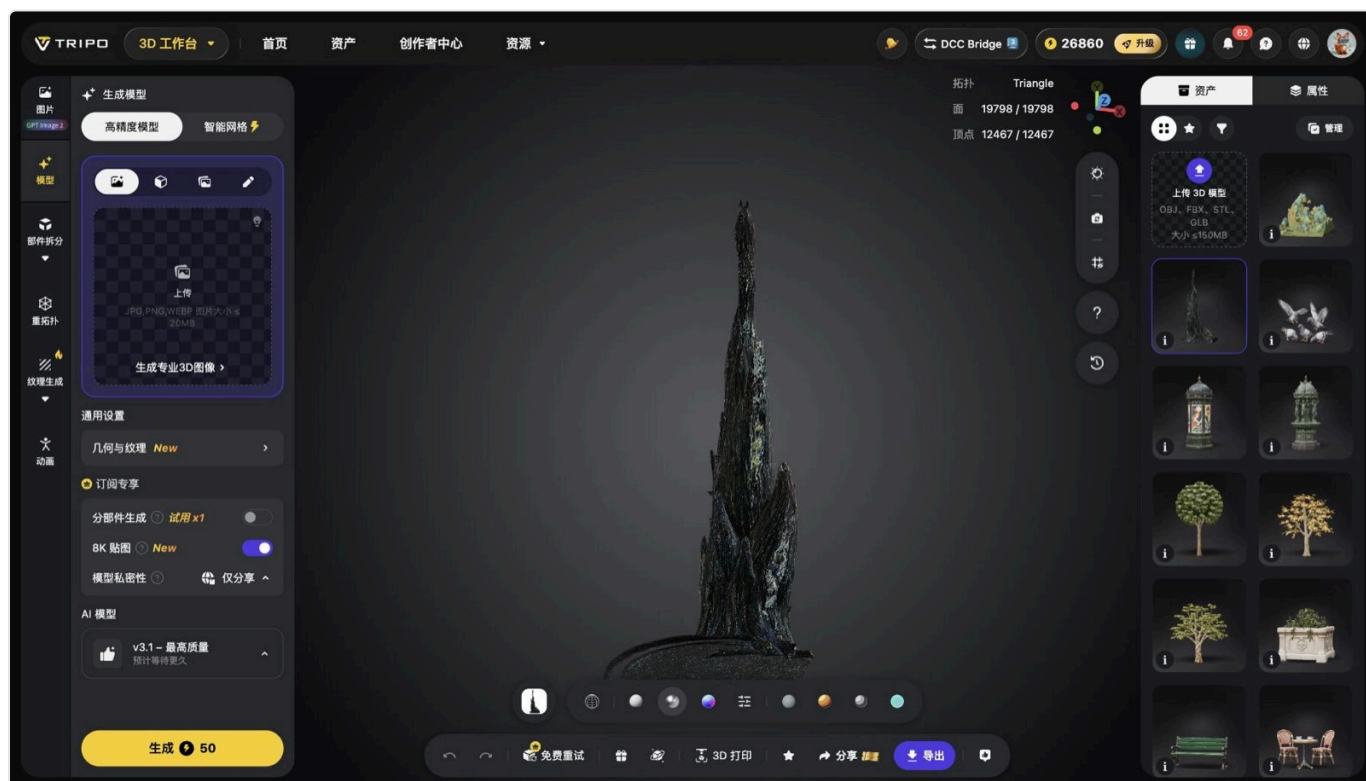
14 · 另一个翻车：梵高的柏树，被理解成了岩石尖塔

这一件没有提示词，输入是从《星月夜》3840 px原图上手工抠出来的柏树，去背景，白底1024×1024。起因是2026-09-10看到@HoodyLiu那条把星月夜做成可拆分3D油画的推文

(<https://x.com/HoodyLiu/status/2097536045881704688>), 想验证Tripo能不能把其中一层做成真的立体。



输入图：星月夜的柏树，手工抠图，白底1024×1024。原作早已进入公有领域，但我抠图用的那份高清复制品，扫描件本身的版权没核过，所以这一件没有进资源包 模型/ 那六组可拿来用的素材（口径见 \$ 00）。



Tripo结果，v3.1最高质量，8K贴图，50积分，约4分钟。面19798/顶点12467。Tripo自动命名：rocky spire 3d model。

两件事同时成立。**好消息：风格保真度不是问题。**油画笔触和颜色被原样烘进贴图，凑近看，梵高那种一道一道的笔触全在，侧视也能确认它是有真实体积的，不是一张竖着的纸。

另一半：形体语义漂移了。那些火焰般向上卷的枝叶被整体理解成了一块岩石尖塔。原因不神秘，§02那条翻车已经拆过：这张输入图本身就不写实，它是一幅后印象派油画的局部，轮廓由笔触构成，没有一条边是硬边。

所以这一条不该记成「它做不了艺术品」，该记成一条可以提前判断的规律：

输入越不写实，形体漂移的概率越高。判据是「这张图上的东西，换个角度看你能不能想象出它长什么样」。你自己都想象不出来，它也补不出来。

怎么避开，两条：一，把这类输入当成「拿贴图」而不是「拿形体」用，我最后就是这么处理星月夜那个demo的，柏树的形体在引擎里重做，Tripo那件只贡献表面；二，如果非要它的形体，先自己画一张写实的设计图，再让它建：绕回§08那条流水线，中间那张图就是干这个用的。

还有一点值得记：它漂移得很诚实。名字直接告诉了你它当成石头了，你第一时间就能发现，而不是等摆进场景才懵。§08那条「读名字当验收」说的就是这个。

核心建议

想自己复现这个实验，不必去找梵高。让出图模型画一张后印象派笔触、轮廓全是软边的测试图送进去就行——上面那条规律只跟「写不写实」有关，跟画的是谁无关。我这一件的输入是名画高清复制品的裁切，扫描件版权没核过，所以它只作为 demo/星月夜 那个成品程序自带的资产存在，没有进 模型/ 那六组素材（§00那四类剔除项里的第二类说的就是它，两层口径在那一节分得很清楚）。

15 · 二次修正的写法，以及一次没必要的返工

第一批跑完，我担心两件东西：玫瑰那根细长花茎、树苗那根细细的芽茎。细长结构在image-to-3D里会不会断掉？我当时认定会，于是立刻写了一个 `jobs-retry.jsonl`，把这两件重画成「粗壮版」。

.....形体概括、块面分明，画面里没有任何细长结构、没有复杂纹理、没有高光渐变，适合 image-to-3D 重建。.....画面内容：一朵盛开的玫瑰，花头又大又饱满、占满整个主体，花瓣厚实、层层叠叠向外张开；花头正下方只有一小截粗壮的绿色花萼，不画长花茎、不画叶子。

.....画面内容：一株猴面包树幼苗，短短的一截粗茎，顶端三片又大又厚、圆钝的叶子向外张开，叶片宽大饱满，整体像一个矮胖的三叶小盆栽。



重做版设计图 01b-玫瑰-短茎大花头.png 与 02b-猴面包树苗-短粗.png，2026-09-10 22:11与22:13。

这两张图最后一张都没送进Tripo。因为原版重建得就是对的。那朵带细长花茎的玫瑰，面19612的模型上，花头、花萼、花茎、叶片一样不缺（\$ 08那张预览图就是它）。

所以我在经验库里写下的那条「细长结构是image-to-3D的弱项」，**没有证据，已经撤回**。这两张图我还是留在这里，因为这两条提示词本身是对的写法：把「短茎大花头」「短粗」这种比例意图写死，把「不画长花茎、不画叶子」这种减法写死，真遇到细长件翻车的时候，这就是修法。只是这次它没派上用场。

核心建议

写完一条修正提示词，先别急着花那50积分。回去看一眼原版的Tripo预览。很多时候你要修的是自己的想象，不是模型。

16 · 把「适合image-to-3D重建」这句话本身写进去

两批提示词的风格段落末尾都挂着这么一句：小王子那批是「适合image-to-3D重建」，巴黎那批是「细节清晰适合image-to-3D重建」，小王子本人那张是「适合image-to-3D重建与自动骨骼绑定」。

这句话不是写给人看的注释。它告诉出图模型「这张图的下游是什么」，模型会因此自己收敛掉一堆不该有的东西：戏剧光、浅景深、局部虚化、艺术性的裁切。**这是整段提示词里最划算的一句**，因为它替你挡掉的是你还没想到要挡的东西。

17 · 「低多边形」是产量词，「水彩插画」是气质词，两个都要给

对照两批的风格前缀就能看出差别。巴黎那批第一个词是「低多边形写实卡通风游戏资产原画」。低多边形（low poly，用尽量少的面构成形体的游戏美术风格）说的是几何复杂度，它让23件资产的面数和细节密度落在同一个量级上，摆进同一条街才协调。

小王子那批第一句是「深色钢笔线条勾出干净轮廓，内部平涂低饱和水彩色块」，说的是气质。但请注意它紧接着还写了「形体概括、块面分明」，这半句干的是和「低多边形」同一件事。

一段好的风格前缀，气质和几何复杂度这两件事要分别说清楚，缺哪个都会出问题：只说气质，16件的精细程度会飘；只说低多边形，出来是一批没有性格的灰积木。

18 • 拒绝清单比形容词管用

把上面十八条里所有的「不」拉出来看，是一张很长的单子：无地面投影、无文字、无水印、没有复杂纹理、没有高光渐变、没有细长垂挂物、不要长出飘带、不画长花茎、不画叶子、不要有天空云水人物建筑文字水印边框坐标网格、无阴影、不裁切不被边框切断、不写任何编号与徽章标识。

我的经验是，写3D参考图的提示词，拒绝清单的长度大概能占到三分之一，而且它比正面形容词稳定得多。「干净」「简洁」「适合建模」这类词每个模型的理解都不一样；「无地面投影」只有一种理解。

唯一要记住的边界是第12条说的那件事：拒绝清单管得住背景和氛围，管不住主体自身该有什么。主体上的东西要正面点名。



十八条技巧的合力：这张画面里，王座来自第02条那套风格前缀，小王子来自第05条那条绑骨提示词，两件东西是分两批、隔了一小时生成的，摆在一起看不出接缝。2026-09-11自检截图。

把这十八条收成一张单子

写完自己的提示词，对着这张单子过一遍，能过就发：

发之前自检十条

- ①四条配方在不在（一个物体、白底、四分之三视角、完整入画）
- ②风格段落和这批别的任务是不是逐字相同
- ③有没有一个真实参考名或者行业术语，而不是一串形容词
- ④量词换成主体自己的了吗（独栋建筑/单棵树/单个角色）
- ⑤比例有没有写（尤其是细长和高瘦的东西）
- ⑥数量有没有数出来，别写「一群」「几个」
- ⑦要绑骨的话，四肢和躯干分开了吗、全身在画面里吗
- ⑧该留白的招牌位置，有没有正面点名说「这里不写字」
- ⑨末尾有没有挂「适合image-to-3D重建」
- ⑩拒绝清单有没有占到三分之一

下一节讲怎么把这十八条批量跑起来：jobs.jsonl怎么写、为什么绝对不能并发、网页端一次提交几件才不会超时、以及那个绿色的勾为什么有时候是骗你的。

§10 批量出图与命名

Batch Generation, Naming, Retries

把上一节那十八条变成一个能跑的批次：jobs.jsonl怎么写、为什么必须串行、命名怎么定才能一路对到GLB、重试放哪、验收该看哪块屏幕。后半节是2026-09-12那次网页端批量重做的实战记录：一批塞几件不会超时、页面退化了怎么自己修回来、十几个文件怎么下载。附一个必须知道的坑：脚本跑的那个绿勾会骗人。

一行一个任务，四个字段

批量出设计图用的是 `gen_via_codex.py`，任务写在一个JSONL文件里（每行一个独立的JSON对象，不是一个大数组）。一行长这样，这是巴黎那批的第一条，原样抄自 `tripo-source/jobs.jsonl`：

```
{"prompt": "风格：低多边形写实卡通风游戏资产原画.....画面内容：一栋巴黎奥斯曼风格转角公寓建筑，弧形立面转角，浅灰色石材外墙，底层拱形商铺橱窗，上层带铸铁阳台栏杆，顶部灰蓝色锌板孟莎屋顶。", "out": "building-01-corner.png", "size": "1024x1024", "quality": "high"}
```

四个字段：`prompt` 是上一节写好的那段，`out` 是输出路径，`size` 是尺寸，`quality` 固定 `high`（中低档会丢细节，而细节正是Tripo用来判断形体的东西）。

跑起来一条命令：

```
python3 gen_via_codex.py --batch jobs.jsonl
```

它走的是编程Agent的订阅额度，不需要API key，跑一批不额外花钱。失败的任务打到错误输出但不中断其余，所以一批里坏一条不会毁掉整批。

绝对不要加--concurrency

这个脚本默认并发是1，也就是串行，一张跑完再跑下一张。不要手贱去调高它。

原因是兜底逻辑：Agent不一定会把图写到你指定的 `out`，脚本的兜底做法是给共享的生成目录拍个快照，跑完求差集，取最新那张搬过去。一并发，多个任务的产物混在同一个差集里互相抢——2026-06-18我实测过，背靠背跑两张，输出的两个文件md5完全相同。

串行的代价其实不大。小王子第一组9张，21:57开跑，22:07收工，十分钟。巴黎两批31张设计图，也是一个下午就跑完了。真正的时间花在挑哪些资产要做上面，不在等图。

注意

那个绿色的勾会骗你。脚本只检查 `out` 这个文件存不存在，存在就打✅。当Agent拿不到生图能力时（内置生图不可用、或者审批策略挡住了），它会自己用SVG加ImageMagick把图「拼」出来写到那个路径，脚本照样打✅。2026-09-03我被这样骗过一次，五张全是合成的。

验真看三个正面证据，任何一条为假就是假图：

1 生成目录有没有新增文件

真图一定会在Agent的生成目录里留一份原件，假图不会。

2 PNG的info是不是空的

`python3 -c "from PIL import Image; print(Image.open('图.png').info)"`，真图是空 `{}`，ImageMagick拼出来的会带一串 `date:create`。别用 `identify -verbose`，那个对真图也会现算一个出来。

3 工作目录有没有残留中间件

冒出 `*.svg` 或者 `paper.png` / `depth.png` / `base.png` 这类文件，基本就实锤了。

这一条我放在这么靠前的位置，是因为它的危害有滞后性：假图长得也像回事，你把它送进Tripo、花掉50积分、拿回一个奇怪的模型，然后开始怀疑Tripo。批量跑完先抽验两张，再往下游送。

命名：设计图叫什么，GLB就叫什么



设计图 `building-05-kiosque.png`，对应的模型文件是 `building-05-kiosque.glb`。前缀 `building-` / `car-` / `pedestrian-` / `prop-` / `tree-` 直接就是资产分类，两位数字是批内序号。

素材/配图/tripo-source/

```
├── jobs.jsonl
├── building-01-corner.png      → paris-demo/assets/building-01-corner.glb
├── building-05-kiosque.png    → paris-demo/assets/building-05-kiosque.glb
├── car-01-taxi.png           → paris-demo/assets/car-01-taxi.glb
└── pedestrian-05-painter.png → paris-demo/assets/pedestrian-05-painter.glb
```

规则只有一条：**同名，只换扩展名**。听起来是废话，但它换来的东西很实在：代码里写一句 `assets/${id}.glb` 就能装配整座城；漏了哪一件，两边目录 `ls` 一对就知道；三个月后回头看，你还能从模型名反查出它是哪条提示词生的。

小王子那批我用的是另一套：设计图叫 `01-玫瑰.png`（两位序号加中文名），模型叫 `rose.glb`（英文短名，因为它要进JavaScript代码）。这套是妥协过的，不如巴黎那套好用，每次都要在脑子里做一次中英对照。要是重来一次，我会全用巴黎那套。

注意

千万别拿Tripo自动起的那个名字当文件名。它是每次生成现写的一句英文描述，同一张图跑两次会给你两句不一样的话（同一张凯旋门的图，一次叫 `arc de triomphe`，一次叫 `grand arch`）。更坑的是它会起得很像另一样东西：我那两棵梧桐，绿冠那件被叫成 `maple tree`、秋黄那件被叫成 `oak tree`，三个月后按名字认，十有八九装反。这个名字的正确用法是当验收看一眼（§08），不是当身份用。**文件名永远由你自己定，而且和设计图同名。**

重试写进一个新文件，不要改原文件



`jobs-retry.jsonl` 跑出来的 `01b-玫瑰-短茎大花头.png`，2026-09-10 22:11。命名带 `b` 后缀，和原版 `01-玫瑰.png` 并排放着，谁也不覆盖谁（另一条 `02b` 和它的提示词在 § 09第15条）。

小王子那批一共有五个任务文件：`jobs.jsonl`（第一组9件）、`jobs-retry.jsonl`（两件重做）、`jobs-prince.jsonl`（主角）、`jobs-planet.jsonl`（两张星球贴图）、`jobs-props.jsonl`（七件道具）。

拆成五个文件有两个好处。一是每个文件都是一份可复现的记录：三个月后我还能一行一行读出当时想要什么；要是在原文件上改，历史就没了。二是重试的成本变得很低：只改那一条、跑一个两行的文件，不用担心误触发另外14条重跑（重跑设计图不花钱，但会白等十分钟）。

重试文件用 `b` 后缀，不用 `-v2`、不用日期。理由是它和原版在同一个目录里按文件名排序时会紧挨着，`01`、`01b`、`02`、`02b`，一眼能看出哪几件返过工。



第二批扩充的 car-03-delivery-van.png。第二批（tripo-source-2/）12件是隔了一天加的树、街具和更多行人，用的是同一段风格前缀。批次可以分开，风格前缀不能改，改了新旧两批就对不上了。

送进Tripo这一段是纯机械活，只有一件事要盯

首页生成表单的样子见 § 01。设计图拖进去，模式选高精度模型，点生成；按钮上那个积分数会随加项开关变。

我在浏览器自动化的笔记里给这一段下过一句评语：批量生成本身是机械活，设计图（Agent出）→ 上传 → 选 v3.1高精度 → 提交 → 轮询积分到账 → 拿接口取GLB。7个连着跑没有额外的坑。

要盯的只有积分余额，它是唯一可信的提交成功信号。

做星月夜那次我在这儿栽过：点完生成，积分没立刻变、页面也没跳转，我判定失败又点了一次，结果两条任务同时在跑，白花50积分。那一轮总共-200积分跑了5次生成，多出来的那50就是这么来的。

核心建议

点完生成后积分没立刻变、URL没跳转，不等于失败。等几秒再复查，不要立刻重试。

09-12那批：网页端连着提交十几件，四条实操

上面那句「纯机械活」是2026-09-10写的，当时一批只有7件。9月12号我为了给视频重做素材，在网页端一口气跑了14次生成加13次贴图生成，这才发现连着提交和提交一件是两回事。四条，按踩到的顺序写。



那一批的产出，2026-09-12。每件下面是文件名和三角面数（12k到29k）。从13:39跑到13:54，网格阶段12件用了约15分钟；加上后面补做的两件和13次贴图生成，这一批实付约1320积分。

1 每提交一件，必须回到生成页重新开一次

提交完一件，页面会留在结果视图上。这时候如果直接接着传下一张图，会失败。那个图片输入框已经被前端清掉了，你以为在上传，其实文件根本没进去。**正确做法是每件之间导航**

回 `/zh/workspace/generate`，从干净的表单重新开始。我一开始图省事想连着传，白白浪费了两轮。

2 回来那一下页面是「半醒」的，上传图片会自己修好

刚导航回生成页的时候，界面会是一个退化状态：按钮上的积分数不对、模式选择器里那个P2.0的选项干脆不见了。别去点，也别刷新，先把图片拖进去。图一上传，整个表单就活过来了，选择器回来了，按钮上的数也对了。我第一次遇到还以为账号出问题了。

3 一批最多塞两件，塞三件会超时

我是让agent替我点的，一个指令里塞3件（12个操作步骤）跑到113秒被掐断，**每批2件稳定跑完**。超时之后最要命的不是慢，是你不知道断在哪一件：有的已经提交了，有的没有。

超时了别重试，先盘账。重试的代价是重复提交，每一次就是65积分。正确做法是去查每件的详情，看它的算子记录里有没有这次的任务，有就是真提交了。我那轮唯一浪费的65积分不是超时造成的，是另一个参数没设对导致的重做（那条坑在 § 01）。

4 下载链接不要在终端里手打

拿到手的模型下载地址是带签名的长链接，里面有 `&` 和 `~`。这两个字符在shell里有特殊含义：`&` 会把命令切成两截丢进后台，`~` 会被展开成你的用户目录。手打一定错，而且错得莫名其妙。**把链接交给Python去下，别经过shell**。13件串行下完大约20分钟，单件8到16 MB、一两分钟一件；中间有一件撞上连接中断，重跑就好，所以下载脚本要写成可以重复跑的（已经下好的跳过）。

这四条的共同点：批量的成本不在「跑得慢」，在「你不知道跑到哪儿了」。所以我现在的习惯是每件之间留一个可查的锚点：回生成页、记一行台账、下完立刻核一次文件大小。一批十几件，多花的这点时间远小于盘一次糊涂账。

验收先看Tripo自己的预览，不看自己引擎里的截图

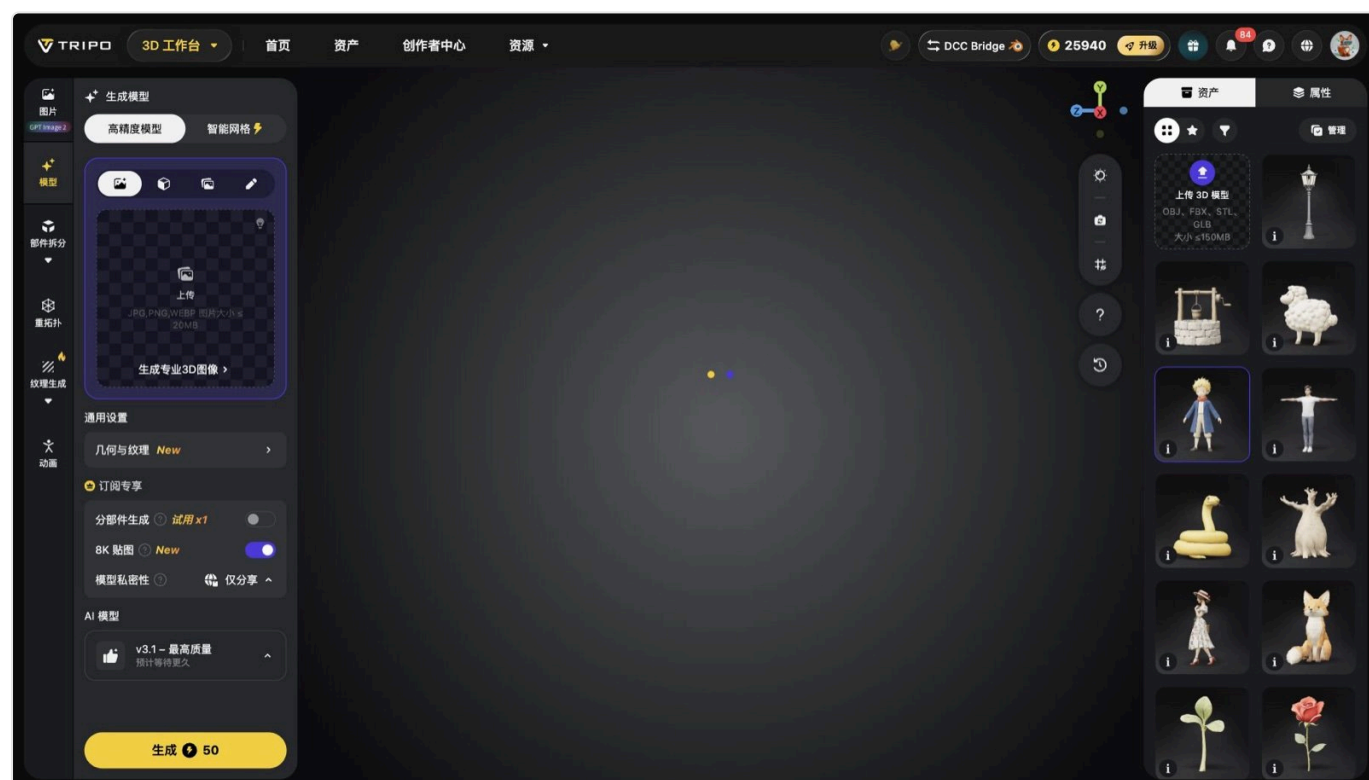
这条我2026-09-10栽得挺惨。

当时小王子的demo已经跑起来了，我在夜景、低角度、贴得很近的截图里看到玫瑰是「一团皱巴巴的红块」、狐狸是「摊在地上的一坨」，直接认定这两件重建失败了，还顺手在经验库里写了一条「细长结构是image-to-3D的弱项」。

两个判断都是错的。去Tripo的任务详情页一看：玫瑰是完整的玫瑰，花头、花萼、花茎、叶片都在；狐狸是完美的坐姿狐狸，耳朵、黑鼻、白胸、尾巴都清楚（那两张预览图在 § 08和 § 09）。

错在读图的方式，不在模型。低多边形模型在夜景低角度近距离下会退化成剪影，红玫瑰缩成一个红团，本来就该这样。而Tripo的预览是免费、直接、权威的一手证据，任务详情页一进去就是它，我却跳过了它去猜。

资产页见 § 03。顶部筛选按批次属性分：All / Smart Mesh / Untextured / Textured / Rigged，批量做完之后靠它点数。



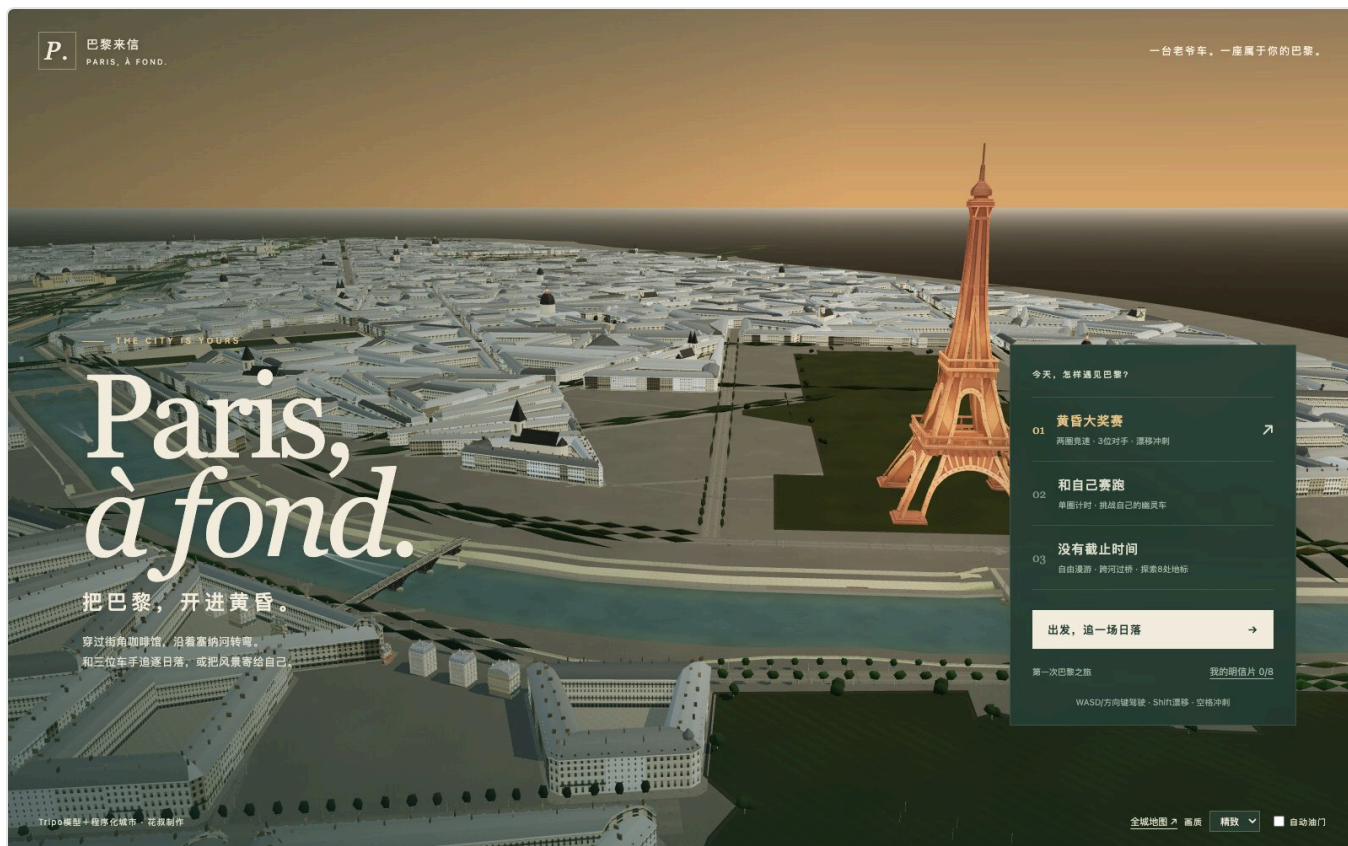
工作台右侧的资产栏，2026-09-10（主视口那一刻正在加载模型，所以是空的，这张图的价值在右边那一列）。一整批缩略图排在一起看，是判断「风格统不统一」最快的办法：十几件东西的明度、饱和度、概括程度是不是一套语言，扫一眼就知道。

所以我的验收顺序固定成三步：**读Tripo自动起的名字（§ 08）→ 看Tripo自己的预览 → 才轮到自己引擎里的截图**，而且引擎截图必须是中性光、正面、镜头退到能看见整体。为了这个我专门写了一个自检脚本，用无头浏览器逐件把摄像机摆到固定位置出图，就是 § 08那张道具拼图。

一批下来的账

	小王子星球	巴黎来信
设计图	22张（21个任务 · 5个jobs文件）	31张（2个jobs文件）
设计图花 费	0（订阅额度）	0（订阅额度）
送进Tripo	16次 × 50积分	23件 × 40积分 = 920 （按单价反推）；开发笔记当时记的是700-800
绑骨	20积分 × 2次（第一次失败要重 试）	那一轮没做绑骨
小计	840积分	按单价反推约920 （未逐笔记账）

两栏的单价对不上，当时我以为是记错了，后来才搞清是档位不同。巴黎那份开发笔记写于2026-09-09，账本上是实打实的40（2190生成一棵树之后剩2150）；小王子那批按钮上稳定是50。同一个「高精度模型」，几何底价15是固定的，差的全在贴图那一档：2K是30、4K是40、8K是50，再把Ultra网格打开就是65。**按这条阶梯反推，40对应的是4K贴图档，50是默认的8K。**阶梯是2026-09-11在表单上一档一档点出来的，完整拆法和我这次的实付明细在 S 07和附录C。巴黎那栏的700-800是开发笔记当时随手写的估数，不是逐笔加出来的：**按23件 × 40反推应该是920，中间这120到220分的缺口我核不出来——那批当时没有逐条余额读数兜底，事后补不回去。**所以这一栏我按附录C那条规矩处理：**能闭合的单独说，不能闭合的标明是哪种数**，小计那一格写的是反推值920，不是那个估数。小王子那栏的840反过来是能闭合的（16 × 50加两次绑骨20）。



这条流水线的终点：31张设计图、23件Tripo模型，加一层程序化城市，变成一个能在浏览器里开车的巴黎。左下角那行小字是我自己标的：「Tripo模型+程序化城市」，哪一层是谁做的，写在页面上。

Part II到这里结束。接下来Part III换一个方向：这些GLB拿到手之后，怎么接进Three.js、Blender、Unity，以及哪一层该交给谁。

§11 把Tripo接进你的工具：五条路的入口与门槛

Five Integration Paths — Entry Points and Blockers

五条把Tripo接进自己工作流的路。其中两条我真的从零装到登录完，连日志带截图；MCP那条补跑了协议握手和工具表，还把它配进Claude Code验到连上；剩下两条给你官方口径和入口在哪。标题写的是「入口与门槛」不是「跑通」：五条路里没有一条走到「一件资产落盘」那一步，缺的是什么、为什么缺，节末单独有一段交代。

注意

先把这一节的口径分清楚，一共三类。Tripo CLI和Codex插件这两条，2026-09-11下午我在自己机器上从零走了一遍，装了、登录了、体检了、跑到发起生成那一步，下面的终端输出和耗时全是实测。MCP是第三类里的半条：CLI自带的 `tripo mcp` 我在2026-09-13补跑了协议握手、工具表和两次读类调用，也把它配进Claude Code、用 `claude mcp list` 验到 `Connected`，那一段是实测；GitHub上那个独立仓库没装没跑。**API和DCC Bridge**这两条我没有接过，写的是官方文档原文加上我在界面上看到的入口。哪一句属于哪一类，我一路标清楚，别混着读。

五条路其实是两类

第一次看Tripo开发者文档，左边导航里躺着CLI、Codex插件、ComfyUI节点、Quick Start、一大串API端点，还有博客里的DCC Bridge和一个GitHub上的MCP仓库。看起来是六七七个并列的选项，很容易挑花眼。

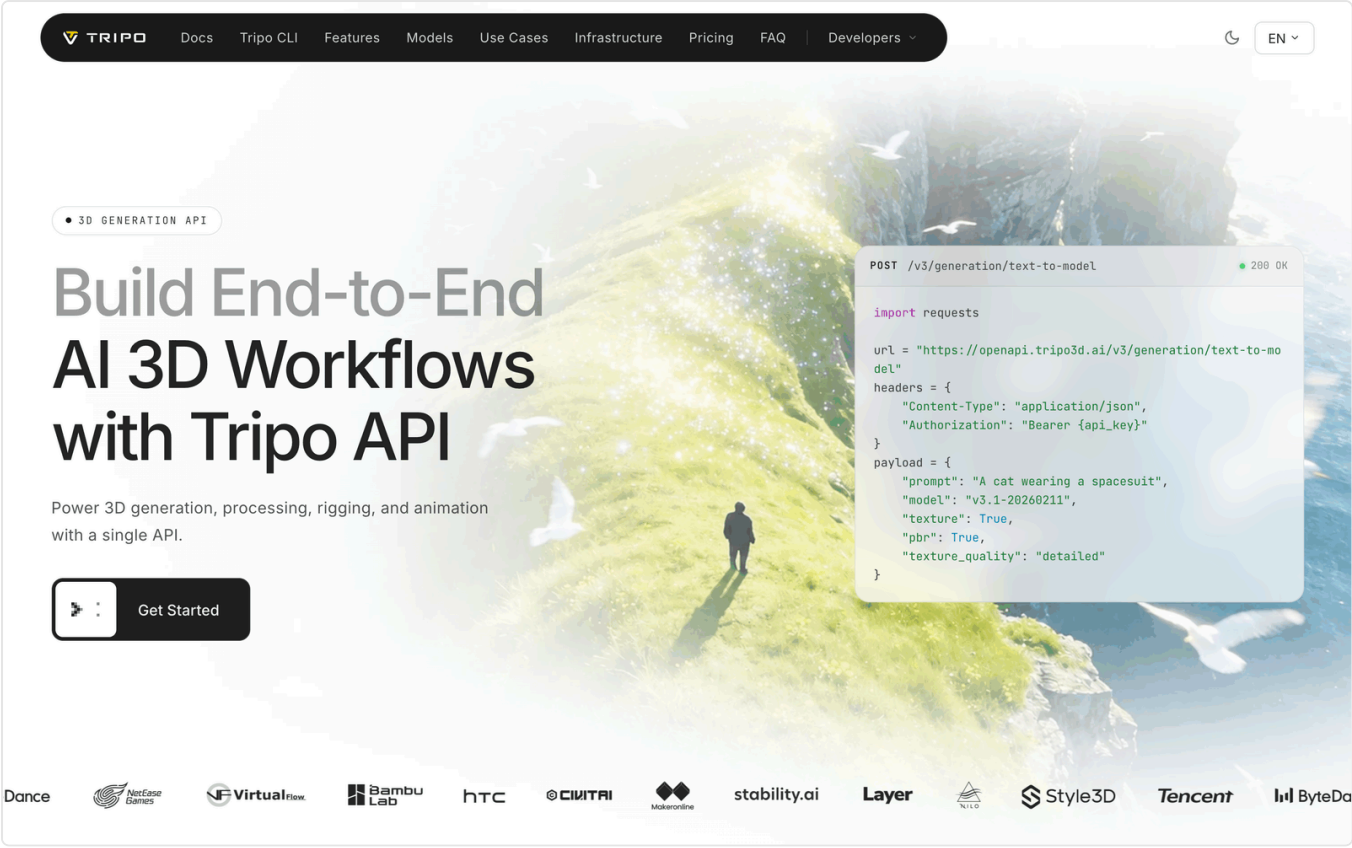
其实只有两类。

第一类是「让AI替你调用生成能力」：Codex插件、CLI、MCP。你在编程Agent里用人话说「给我做个宝箱放到assets目录」，它自己去调、自己下载、自己放进你的项目。这一类的门槛是Node和一次浏览器登录，不需要你懂API。

第二类是「把模型送进你的软件」：DCC Bridge、以及API的下载环节。模型是你自己在网页上生成的，这两条只负责把文件搬到该去的地方。

核心建议

动手之前先确认机器上有**Node**。CLI和Codex插件这两条路都靠它，官方要求是**Node.js 20或更新**。终端里敲 `node -v`，打得出 `v20.x` 以上就行；提示 `command not found` 就先去 `nodejs.org` 下当前LTS装一遍，macOS也可以 `brew install node`，装完重开一个终端窗口再验。本书里凡是 **npm** 或 **npx** 开头的命令，都以这一步为前提，§04、§12、§13、§14的降贴图命令也算在内。



developers.tripo3d.com首页，2026-09-11。裸域会302跳到中文版。首屏那段代码就是全部API的样子：POST一个端点、带Bearer token。页面中部「一条指令，让Agent接入Tripo」那块，是整个vibe coding路线的入口。

路径	我接过没有	门槛	什么时候选它
Tripo CLI	接了	低	用Claude Code或Cursor；或者要批量跑
Codex插件	接了	最低	你用Codex，只想说人话
API	没接	高	要接进自己的网站或产品
MCP	跑了握手、工具表、Claude Code配置	中	你的Agent只认MCP协议
DCC Bridge	没接	低，但要订阅	在网页上生成，要一键推进Blender或Unity

按你是谁，先看这一节帮不帮得上你。

你想让Claude Code、Cursor或者Codex替你调生成——这一节的主干就是写给你的：装、登录、体检、配MCP、发起生成，每一步长什么样、卡在哪、退出码怎么读，全在下面。到「发起生成」为止都是我机器上真跑出来的。

你想把Tripo接进自己的网站或产品——那你要的是API那条，而这一节给你的只有官方口径和入口在哪，没有一行是我机器上跑出来的。下面「API」那一小节会把你自己得补的三件事点名，与其在这儿等，不如直接去developers.tripo3d.com的Quick Start照着写，那是唯一的一手来源。

你只想在网页上生成、然后一键送进Blender或Unity——跳到本节最后的DCC Bridge，它和AI Agent没关系，是一条传输通道。

先说结果：两条都被挡住了，而且卡点不是同一个

我得把最重要的一条先讲了，因为它会改变你的预算方式。

CLI那条卡在Tripo的API钱包是0积分。Codex那条卡在我的Codex订阅额度耗尽。两条路都走到了「发起生成」才被拦下，全程Tripo API积分变化是0 → 0，一分没花，也没买。

注意

为什么钱包是0？因为API和Studio订阅是两套账，官方明说不互通。我在开发者平台FAQ里把那条展开了，原文是：「不能。Tripo API与Tripo Studio的计费体系相互独立，需要分别购买。」实测完全对得上：同一个账号，Studio里是专业版订阅、生成按钮照常能点；platform.tripo3d.com/billing上API Wallet是0 / 0 Credit，CLI里 tripo balance 也是0。你在Studio充的钱，一分都用不到CLI和插件上。

TRIPO

文档中心

Tripo CLI

功能

模型

应用场景

基础设施

定价

常见问题

开发者社区

技术规格、计价体系与数据安全策略，答案都在这里。

常见问题

联系我们

business@tripo3d.ai

API 有企业版定价方案吗？

有。针对大规模用量的企业客户，我们提供定制价格与批量折扣，详情请联系 business@tripo3d.ai。

购买的积分有效期多久？

积分永久有效，不会过期。

通过 API 提交的输入数据和生成的模型，会被用于模型训练吗？

不会。我们不会将任何用户数据（包括输入数据和生成模型）用于模型训练。

Tripo API 与 Tripo Studio 的生成效果一致吗？

一致。两者由相同的底层模型与算法驱动，生成质量完全相同。

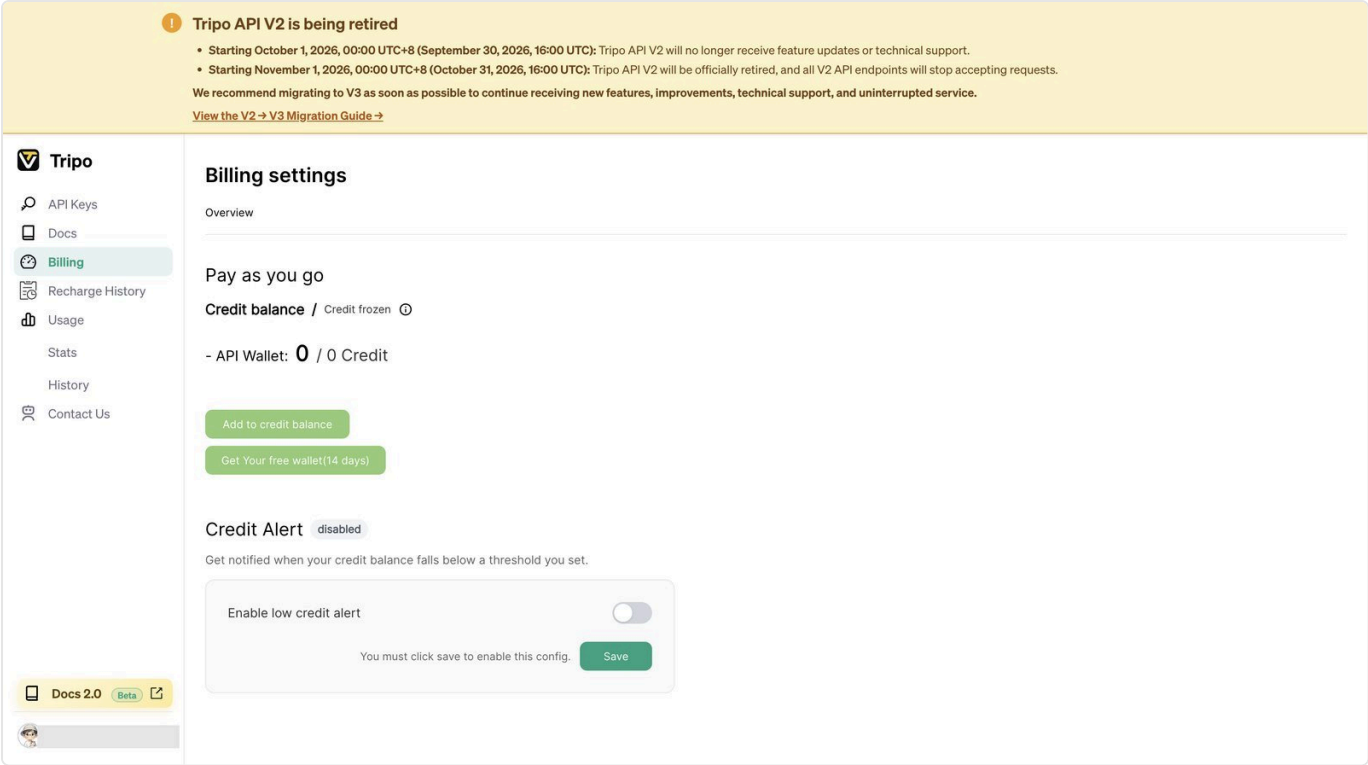
任务失败会扣积分吗？

不会。失败任务不收费；若因系统问题被临时扣除，将自动退还。如需技术支持，请联系 business@tripo3d.ai。

订阅了 Tripo Studio，能直接使用 API 服务吗？

不能。Tripo API 与 Tripo Studio 的计费体系相互独立，需要分别购买。

developers.tripo3d.com首页FAQ展开后的原文，2026-09-11实测。这一条在我第一次采集时是折叠的、答案没取到，这次点开了。



同一个账号在 `platform.tripo3d.com/billing` 上的API Wallet: 0 / 0 Credit。旁边那个「Get Your free wallet(14 days)」按钮是新账号的一次性权益，我没有点它：它带14天倒计时，该留给真正要连着跑测试的那一周。

顺手把另外四条折叠的FAQ也取全了，这五条合起来才是完整的计费图景：

问题	官方答案原文
购买的积分有效期多久？	积分永久有效，不会过期。
输入数据和生成的模型会被用于模型训练吗？	不会。我们不会将任何客户数据（包括输入内容与生成模型）用于模型训练。
API与Studio的生成效果一致吗？	一致。两者由相同的底层模型与算法驱动，生成质量完全相同。
任务失败会扣积分吗？	不会。失败任务不收费；若因系统问题被临时扣除，将自动退还。
订阅了Studio，能直接使用API服务吗？	不能。两者计费体系相互独立，需要分别购买。

注意

还有一条写代码前必须知道：Tripo API V2要下线了。上面那张钱包截图顶部那条黄色横幅就是它：V2自2026年10月1日停止功能更新与技术支持，2026年11月1日正式下线，所有V2端点停止接受请求，官方建议尽快迁到V3。本书所有示例走的都是V3。

路径一实录：Tripo CLI

环境：macOS 26.6.0 / Apple Silicon / Node v26.0.0 / npm 11.12.1。全过程14:04到14:51。

1 安装：一条命令，但第一次要等五分钟

```
npm install -g tripo-cli
# added 11 packages in 5m
```

冷装实测4分49秒，同一台机器第二次跑238毫秒。慢的是国内网络拉registry，不是包大。

注意

这五分钟里终端一个字都不打。npm检测到stdout是管道就把进度条关了，看起来像卡死。它没卡，等着就行。官方文档提到macOS可能撞EACCES要加 `sudo`，我本机Node装在homebrew前缀下、普通用户可写，没需要。

装完验证版本：`tripo` 在 `/opt/homebrew/bin/tripo`，版本0.4.0。

核心建议

截终端做公开物料，一律把标题栏裁掉。macOS的标题栏会把前台进程的环境变量写进去。这一轮的第一张截图就把一个第三方API key拍了进去，只能删图重截。设自定义标题也没用，那只是追加、盖不住。

2 体检：文档说三项，实际是六项

```
tripo doctor
```

```
tripo-lab $ tripo doctor
✓ node v26.0.0 (need >= 20)
✗ api key missing - run "tripo login"
! api reachability skipped (no key) · would use https://openapi.tripo3d.ai (region ov)
✓ llm (tripo ai conversation mode) not configured - wizard mode only (this is fine)
✓ terminal interactive TTY
✓ language auto
tripo-lab $
```

登录前的 `tripo doctor`，2026-09-11实测。官方文档写的是「依次检查登录状态、网络访问和积分余额」三项，实际打出来六项：node版本、api key、api可达性、llm、终端、语言。登录之后会变成八项（多出config权限和余额）。

这时候 `api key` 那行是红叉，`api reachability` 是感叹号「skipped (no key)」，都正常，因为还没登录。

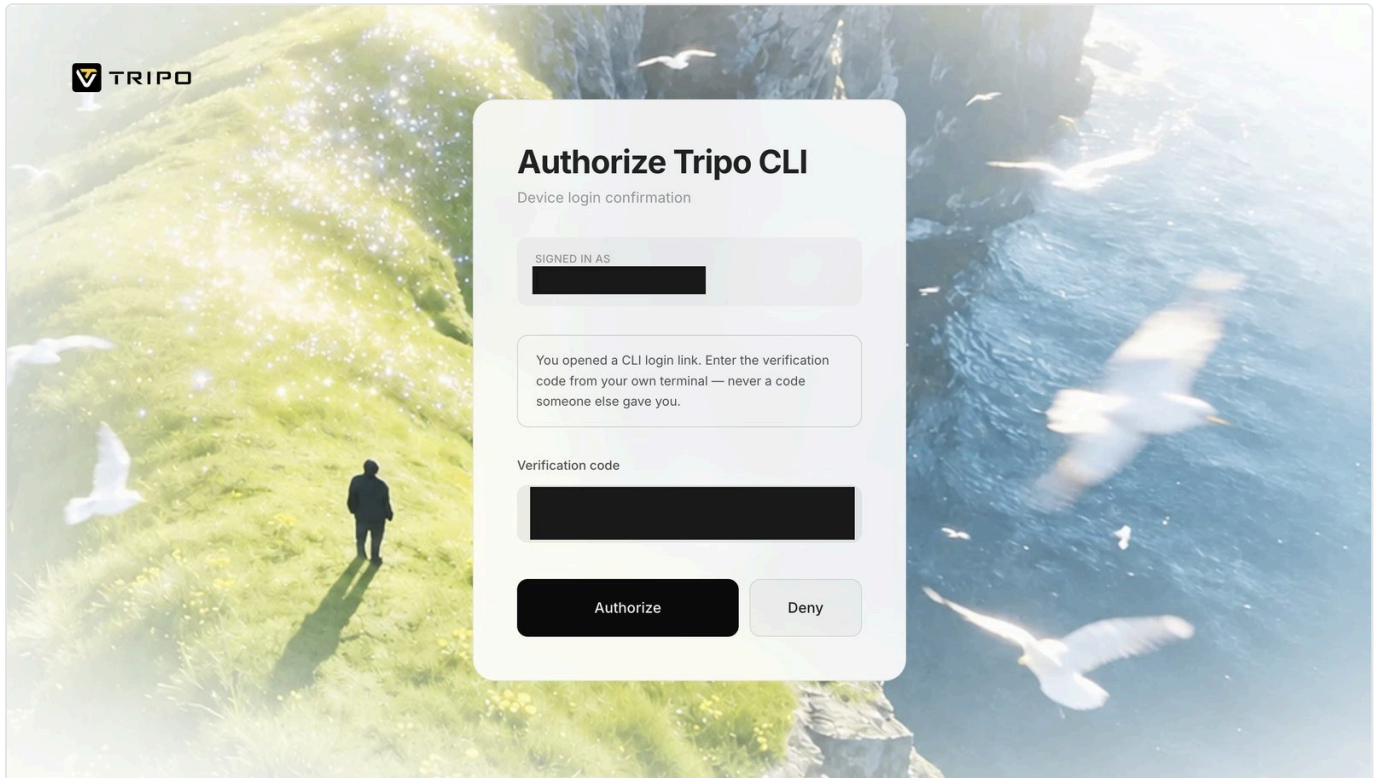
3 登录：设备码加浏览器授权，耗时210秒

```
tripo login
```

终端5秒内打出一个四位加四位的终端码、一个授权网址、以及三步人话说明（打开网址、填终端码、点 Authorize），然后阻塞等你。`--region ov` 是国际区、`cn` 是中国大陆区，不带的话浏览器流程会让你选。从发起到打出「网页授权完成」一共210秒，其中绝大部分是人在浏览器里操作的时间。

注意

这里有一个文档没写、但你一定会撞上的坑：CLI打印的网址带着 `?code=XXXX-XXXX`，但页面会把这个query丢掉，验证码输入框仍然是空的。所以官方文档第2步「在页面填入终端码」不是可选项，是必须手打。这大概率是刻意的，设备码流程的安全性就建立在「人眼比对终端和网页上这两个码」上。



浏览器授权页（验证码与账号邮箱已打码）。另一处和文档对不上：官方文档和插件自带的说明都写「账号已有多个API key时会让你挑一个」，实测页面上没有这一步，只有登录身份、一行安全提示、验证码输入框、Authorize和Deny两个按钮。

授权成功后终端自己往下走，打出的是：

- ✓ 网页授权完成
- ✓ Key 验证通过(region=ov), 当前余额 0 积分
- ! 余额为 0:新账号可在控制台领取免费额度,或运行 `tripo topup` 充值

注意

key是明文存在 `~/.tripo/config.json` 里的（权限0600）。做演示、录屏、截图时别 `cat` 它，也别让 `tripo whoami` 的输出进画面，它会打半截key。

```

tripo-lab $ tripo doctor
✓ node v26.0.0 (need >= 20)
✓ api key from profile:default
✓ config permissions 0600 (expect 0600)
✓ api reachability https://openapi.tripo3d.ai (region ov) · balance 0
! balance 0 credits (0 frozen) - run "tripo topup"
✓ llm (tripo ai conversation mode) not configured - wizard mode only (this is fine)
✓ terminal interactive TTY
✓ language auto
tripo-lab $ tripo whoami
key [REDACTED] (profile:default)
region ov
api https://openapi.tripo3d.ai
余额 0 积分(冻结 0)
tripo-lab $ tripo balance
余额 0 积分(冻结 0)
tripo-lab $

```

登录后的 `tripo doctor` (八项)、`tripo whoami`、`tripo balance`，key已打码。注意第三行 `config permissions 0600 (expect 0600)`，CLI自己会查配置文件权限。余额那一行是 `0 credits (0 frozen)`。

4 生成：被余额预检拦下，退出码4

```
tripo make prop-02-bench.png --model tripo-p2 -p quad=true --no-open
```

```

tripo-lab $ tripo make prop-02-bench.png --model tripo-p2 -p quad=true --no-open
! quad=true forces FBX output (quad topology cannot be stored in GLB)
× 积分不足
_ → 运行 tripo topup 充值
tripo-lab $ echo "EXIT=$?"
EXIT=4
tripo-lab $

```

喂的是一张巴黎长椅的参考图。18秒后被拦下：积分不足 → 运行 `tripo topup` 充值，`EXIT=4`。这正好实测印证了退出码表里那条「4=积分不足，模型还没开始生成，不计费」，余额前后都是0，对得上。

这条命令顺手打出了一个文档里没有的事实： `quad=true forces FBX output (quad topology cannot be stored in GLB)`。**四边面拓扑存不进GLB，会被强制转成FBX。**这同时解释了 § 03里那个现象：智能网格任务详情接口里的 `model_url` 是一个 `.fbx` 而不是 `.glb`。想要四边面就别指望拿到GLB。

还有一条关于模型版本的更正。文档页只提 `tripo-p1` 和 `tripo-v3.1`，说「面数预算两万以内走p1、其余默认v3.1」。但 `tripo docs --topic commands/make` 里还有一个 **`tripo-p2`**，而且它从不被自动选中，必须手动写 `--model tripo-p2`。三个别名映射到的线上版本号是 `v3.1-20260211` / `P1-20260311` / `P2-20260801`。

核心建议

这串版本号和你从接口里读回来的不是同一套写法，别对着找。§ 04那张模式对照表里，智能网格那一列的 `model_version` 读回来是 `Nexus-v2.0-20260801` ——日期段和CLI这边的 `P2-20260801` 完全一样，指的是同一代模型（也就是本书一路叫的「P2.0」），只是开发者文档用 `tripo-p2` 这个别名、接口返回内部名。接口返回里找不到「P2」两个字，不代表你拿错了模型。同理，§ 05里那个跑完纹理就变成 `v3.5-20260815` 的现象，是纹理算子有自己的版本号，和网格算子的版本号本来就不是一个东西。

5 本地预览：不花积分，而且能看任意GLB

```
tripo view car-01-taxi.glb
```



`tripo view` 起了一个本地服务，浏览器里用model-viewer渲染，约6秒出画面。它接受任意本地 `.glb` 路径，不需要账号里有任务、也不需要积分，等于Tripo CLI白送了一个零配置的本地3D预览器。

核心建议

这条是我这次最意外的收获。不管模型从哪来，`tripo view xxx.glb` 就能看，比自己起http server再写个页面省事得多。坑是：预览页跑在后台标签页时WebGL不渲染、截出来是纯黑（和§12那条 `rAF` 被暂停是同一类问题），要截图就把标签页切到前台。

路径二实录：Codex插件

1 装插件：不用开GUI，一条命令14秒

官方文档写的是在Codex里打开Plugins → Plugins Directory → 搜Tripo 3D → Install。实测这一步根本不需要GUI。

```
codex plugin add tripo-3d@openai-curated-remote
```

14秒装完，版本tripo-3d 0.2.1，作者VAST，MIT协议。装完即启用，`codex plugin list` 显示 `installed`, `enabled`，`~/.codex/config.toml` 里不落条目。

```
tripo-lab $ codex plugin add tripo-3d@openai-curated-remote
Added plugin `tripo-3d` from marketplace `openai-curated-remote`.
Installed plugin root: /Users/alchain/.codex/plugins/cache/openai-curated-remote/tripo-3d/0.2.1
tripo-lab $
```

命令行装插件，2026-09-11实测。这是这次实录里我认为最有价值的一条更正：官方文档只给了GUI路径。

把插件包拆开看，它不是MCP server，是两个Agent Skill。目录里是 `skills/tripo-3d/SKILL.md`（通用生成）和 `skills/tripo-game-asset/SKILL.md`（游戏资产管线配方），两份文件里全是shell命令，调的就是路径—那个 tripo CLI，认证读的就是路径一写下的 `~/.tripo/config.json`。SKILL.md原话：Auth resolves from `TRIP0_API_KEY` (a tsk... key) or a previous tripo login.

所以路径二和路径一共用同一个API钱包。这条推论被我自己的失败印证了：路径一余额是0，路径二就算模型额度充足也一样生不出东西。你要为这两条路准备的是同一笔钱。

2 发消息：被Codex订阅额度挡住

```
codex exec -s workspace-write -c sandbox_workspace_write.network_access=true \
  "给我生成一个巴黎长椅的 3D 模型，保存到 ./assets"
```

```
tripo-lab $ codex exec -m gpt-5.5 -c model_reasoning_effort=low -s workspace-write -c sandbox_workspace_write.
network_access=true "给我生成一个巴黎长椅的 3D 模型，保存到 ./assets" 2>&1 | tee ../logs/08-codex-exec-55.log
2026-09-11T06:47:35.645248Z ERROR codex_models_manager::manager: failed to refresh available models: timeout w
aiting for child process to exit
2026-09-11T06:47:35.674275Z ERROR codex_models_manager::manager: failed to refresh available models: timeout w
aiting for child process to exit
OpenAI Codex v0.153.4
-----
workdir: /Users/alchain/Documents/写作/09-实验项目/tripo-接入实录-202609/codex-demo
model: gpt-5.5
provider: openai
approval: never
sandbox: workspace-write [workdir, /tmp, $TMPDIR] (network access enabled)
reasoning effort: low
reasoning summaries: none
session id: 01a08f38-bdfe-7740-8153-e0388081b754
-----
user
给我生成一个巴黎长椅的 3D 模型，保存到 ./assets
warning: Skill descriptions were shortened to fit the skills context budget. Codex can still see every skill,
but some descriptions are shorter. Disable unused skills or plugins to leave more room for the rest.
ERROR: You've hit your usage limit. Visit https://chatgpt.com/codex/settings/usage to purchase more credits or
try again at Sep 15th, 2026 10:25 AM.
ERROR: You've hit your usage limit. Visit https://chatgpt.com/codex/settings/usage to purchase more credits or
try again at Sep 15th, 2026 10:25 AM.
tripo-lab $
```

两次都在模型第一次回复之前就被拒：You've hit your usage limit。换小模型重试同样被拒。./assets 是空目录，一个文件都没落盘。Tripo积分0、Codex积分0，两边都没扣。

核心建议

失败里有一个正面信号值得读出来。第二次重试时Codex打了一句 Skill descriptions were shortened to fit the skills context budget —— 这证明插件的skill确实被装进了这次会话的上下文，挡住的是OpenAI的订阅额度，不是插件没生效。

注意

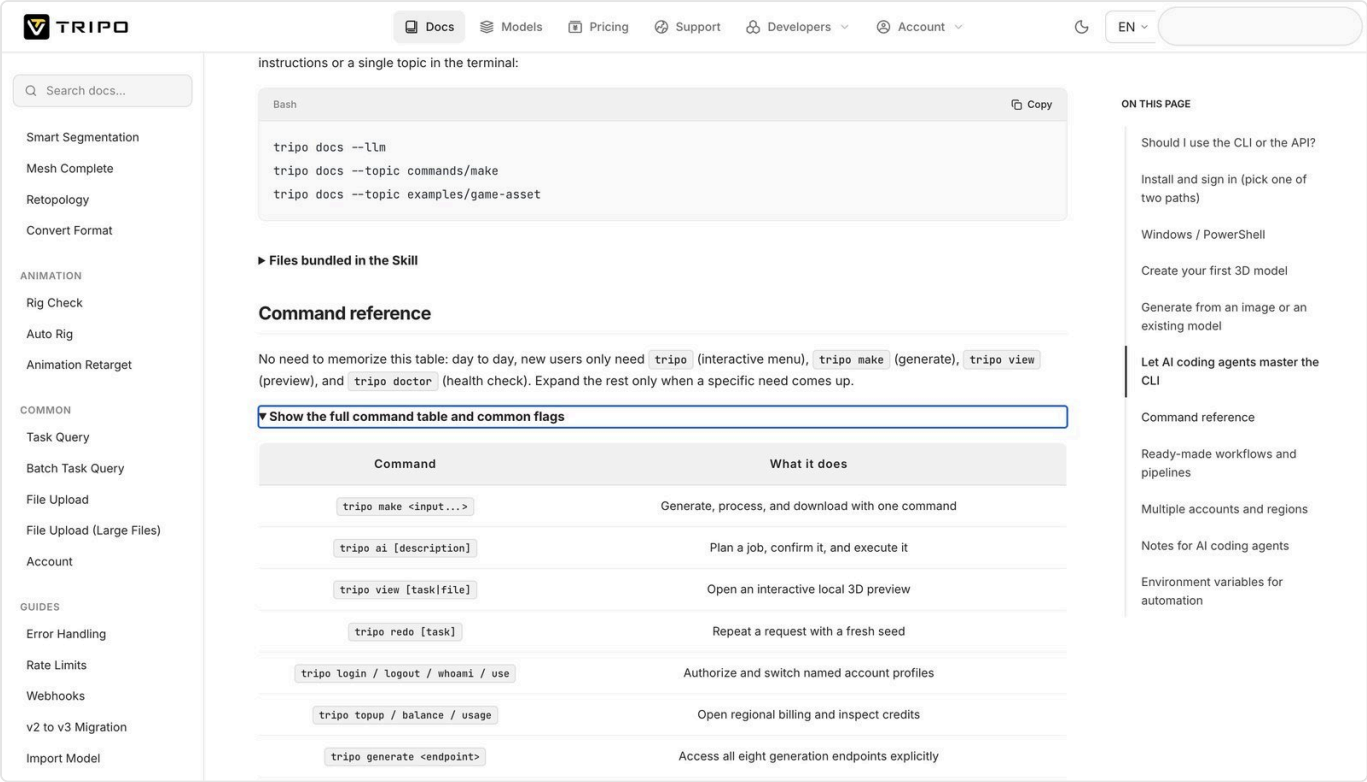
一个和Tripo无关、但会让你查半天的本机坑：一个坏掉的marketplace会让整个插件列表打不开。我本机 `~/.codex/config.toml` 里有一条指向已被清理的临时目录的marketplace，它会让 `codex plugin list` 整条命令报错退出，而不是跳过那一个源。`codex plugin add` 不受影响，它只解析目标marketplace。

文档和实际对不上地方

把这两轮撞到的整理成一张表。左边是文档怎么写，右边是我机器上实际怎样。最后一条属于后面MCP那一小节，一并放这儿。

文档怎么写	实际怎样
授权链接带 <code>?code=</code> ，暗示页面会自动填	页面把query丢了，验证码必须手打
<code>tripo doctor</code> 检查三项	未登录六项、登录后八项
「账号有多个key会让你挑一个」	授权页没有这一步
Codex插件必须在GUI里装	<code>codex plugin add</code> 一条命令，14秒
「装完你会看到两个能力」	它是两个Agent Skill，不是MCP server；和CLI共用一个钱包
只提 <code>tripo-p1</code> / <code>tripo-v3.1</code>	还有 <code>tripo-p2</code> ，且从不自动选中
「新账号可在控制台领取免费额度」	控制台上是「14天试用钱包」，和「积分永久有效」不是一回事
没提 <code>quad</code> 与输出格式的关系	<code>quad=true</code> 强制转FBX
<code>tripo mcp --help</code> 写「tools: make/task/balance/history」四个	<code>tools/list</code> 实际返回五个， <code>task</code> 拆成 <code>task_get</code> 和 <code>task_wait</code>

命令表：日常只用四条



CLI文档里的Command reference，采集时这块是折叠的，手动展开后取的原文。官方自己说日常只需要四条：`tripo`、`tripo make`、`tripo view`、`tripo doctor`。右上角账号区已遮挡。

`tripo make` 能吃图、吃多视图、也能吃已有模型：

```
tripo make concept.png --for print
tripo make front.png back.png
tripo make hero.glb --then texture,rig
tripo make @last --then convert:fbx
```

`--for` 后面跟七个内置预设：`game-mobile`、`game-pc`、`film`、`print`、`ar-web`、`anim`、`toy`。`@last` 是「最近一个任务」，任何接受任务ID的地方都能用。什么都不想记就直接敲 `tripo`，它会开一个交互菜单。

退出码表值得贴在脚本里，因为它把「白花钱」和「没花钱」分得很清楚：

退出码	含义	要不要心疼积分
3	认证错误	文档未说明
4	积分不足	没开始生成，没扣（本次实测印证）
5	内容策略拒绝	文档未说明
6	任务失败	自动退款

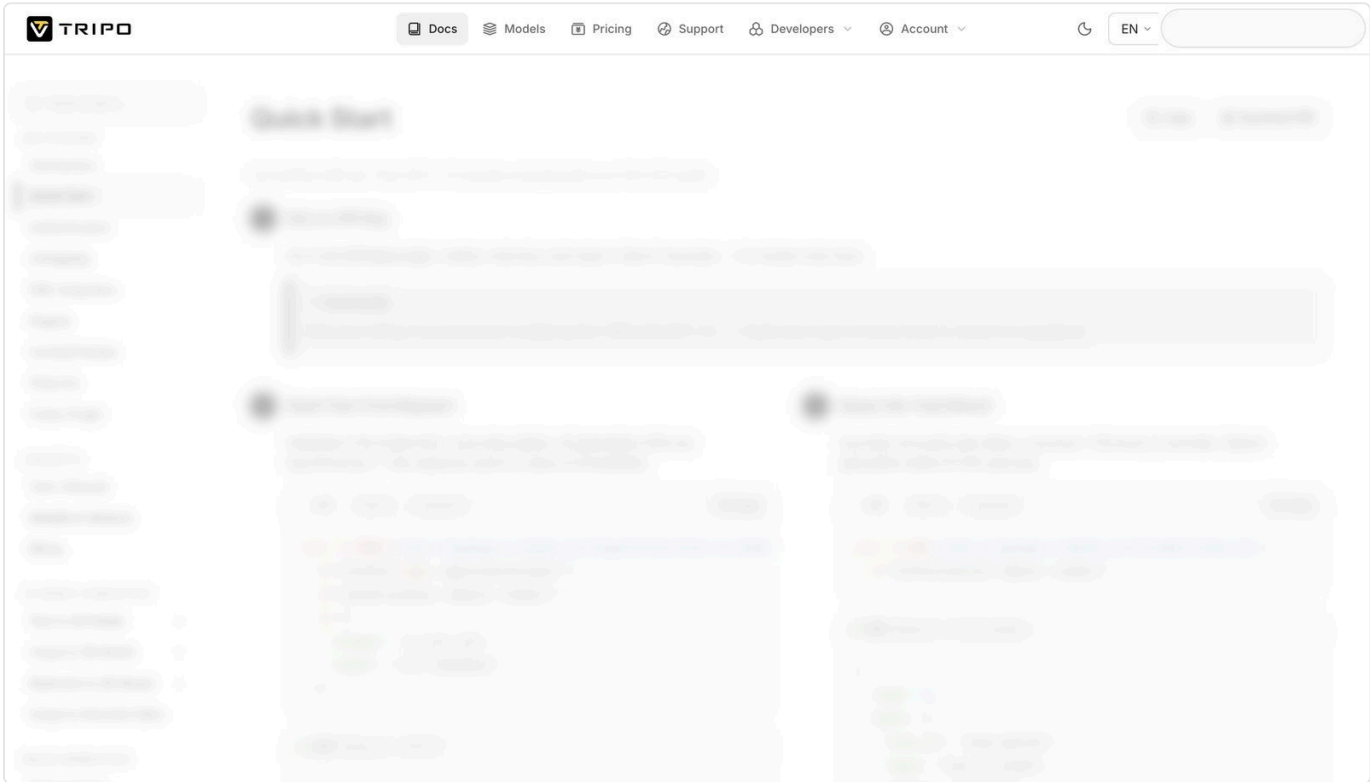
退出码	含义	要不要心疼积分
9	触发限流	退避后重试

Windows用户有三件事要先做（这几条我没有Windows机器可验，是文档原文）：npm装的是一个 `tripo.ps1` 脚本、PowerShell默认会拦住它，跑一次 `Set-ExecutionPolicy -Scope CurrentUser RemoteSigned`；`@last` 和带逗号的参数要加引号；配置和历史在 `%USERPROFILE%\tripo`。

剩下三条路：API、MCP、DCC Bridge

API

API Key在platform.tripod.com的API Keys页创建，只显示一次。基址 `https://openapi.tripod.com/v3`，认证走 `Authorization: Bearer <token>`。所有生成接口都是异步的：POST拿 `task_id`，然后 GET `/v3/tasks/{task_id}` 轮到 `status: "success"`，从 `output.model_url` 下 GLB。官方给的节奏是每2秒轮一次、不要超过每秒1次请求，典型生成10到120秒。



developers.tripod.com/en/docs/quick-start, 2026-09-11采集。右上角账号区已遮挡。

文档里那条Game-Ready Character管线最值得看，它把我在 § 06里手点的绑骨流程写成了四个接口调用：



官方标的预估是约105秒、约85积分，并提醒两条：调rig之前一定先调rig-check，它会返回推荐的骨架类型、避免失败；`retarget` 只吃已绑骨模型的task_id。这和我在 § 06里踩的坑对得上，网页版默认骨架预设是「适

用于动物」、人物必须手动换，走API的话 `rig-check` 把这件事自动化了。

注意

上面这一小段全是官方文档口径，一行都不是我跑出来的，所以我得把你自己要补的三件事点名，免得你读完以为手上有了什么。

一、一次真实的task返回长什么样。我没有一份自己跑出来的 `GET /v3/tasks/{task_id}` 响应JSON，所以 `status` 的中间状态怎么流转、`output` 里除了 `model_url` 还挂着什么、失败时 `code` 和文案怎么配，我给不了你实例。

二、轮询要写成什么样才不会被限流。官方节奏是每2秒一次、每秒不超过1次请求，但退避怎么做、超时判多久、断线重连拿不拿得回同一个task，这些是写产品必须定的，我没有踩过的经验可以给你。

三、签名直链的有效期。`model_url` 是带签名的临时地址，能活多久、过期之后怎么重取，决定了你的服务端要不要转存——§ 03量过Studio那侧的签名链接是约两天过期，但那是另一条通路上的读数，不能直接当API这案的答案。

这三件事我不打算靠转述文档来凑满，因为写产品的人真正会被坑的就是这三处，而转述给不了你任何在这三处上有用的东西。要接产品就直接去Quick Start照着跑一遍，比读我复述的官方文档省时间。

MCP：两条路，而且新旧分裂

第一条是官方CLI内置的 `tripo mcp` 子命令，命令表里写着 `Run the CLI as an MCP server`。你装了CLI就有了它。

第一条我补测了。2026-09-13我用一个自写的stdio客户端跟它走了一遍MCP协议：`initialize`、`tools/list`、两次读类 `tools/call`。这一轮同样是零消耗，没有发起任何生成。

握手返回的是 `protocolVersion 2024-11-05`、`serverInfo` 为 `tripo-cli 0.4.0`，`capabilities` 里只有 `tools`，没有 `resources` 也没有 `prompts`。`tools/list` 列出五个工具：

工具	干什么	必填参数
<code>tripo_make</code>	从文字、图片路径或URL、已有task id生成资产；阻塞到跑完，下载产物并返回本地文件路径	<code>input</code>
<code>tripo_task_get</code>	按id查任务状态与详情	<code>task_id</code>
<code>tripo_task_wait</code>	阻塞等任务跑完，可选下载产物	<code>task_id</code>
<code>tripo_balance</code>	查账户积分余额	无
<code>tripo_history</code>	列这台机器上CLI建过的任务	无

`tripo_make` 的入参和命令行那条 `tripo make` 是一一对应的：`scenario` 就是那七个预设，`then` 就是处理链（原文举例 `"texture,rig,convert:fbx"`），`model` 三选一、`tripo-p2` 的括注写着 `P-series preview with quad`，`output_dir` 是产物目录。换句话说MCP这一层没有新增能力，它是同一套CLI换了个协议门面。

两次读类调用把前面那条钱包推论又往外扩了一格。前面说路径一和路径二共用一个API钱包，是从插件包里的 SKILL.md 推出来的；这次 `tripo_balance` 直接返回 `balance 0 / frozen 0`，和两天前 `tripo balance` 的读数一模一样，**MCP这条也在同一个钱包上，三条路一笔钱**。`tripo_history` 返回空数组，也对得上——这台机器上从来没有一个任务真正建成过。

配进Claude Code的写法是下面这段，2026-09-13实测。它是stdio server，不占端口、不用你自己起进程，把命令交给客户端就行。项目根放一个 `.mcp.json`：

```
{
  "mcpServers": {
    "tripo": {
      "command": "tripo",
      "args": ["mcp", "--quiet", "--no-open"]
    }
  }
}
```

不想写文件的话，在项目目录里敲一条命令让它替你写：

```
claude mcp add --scope local tripo -- tripo mcp --quiet --no-open
```

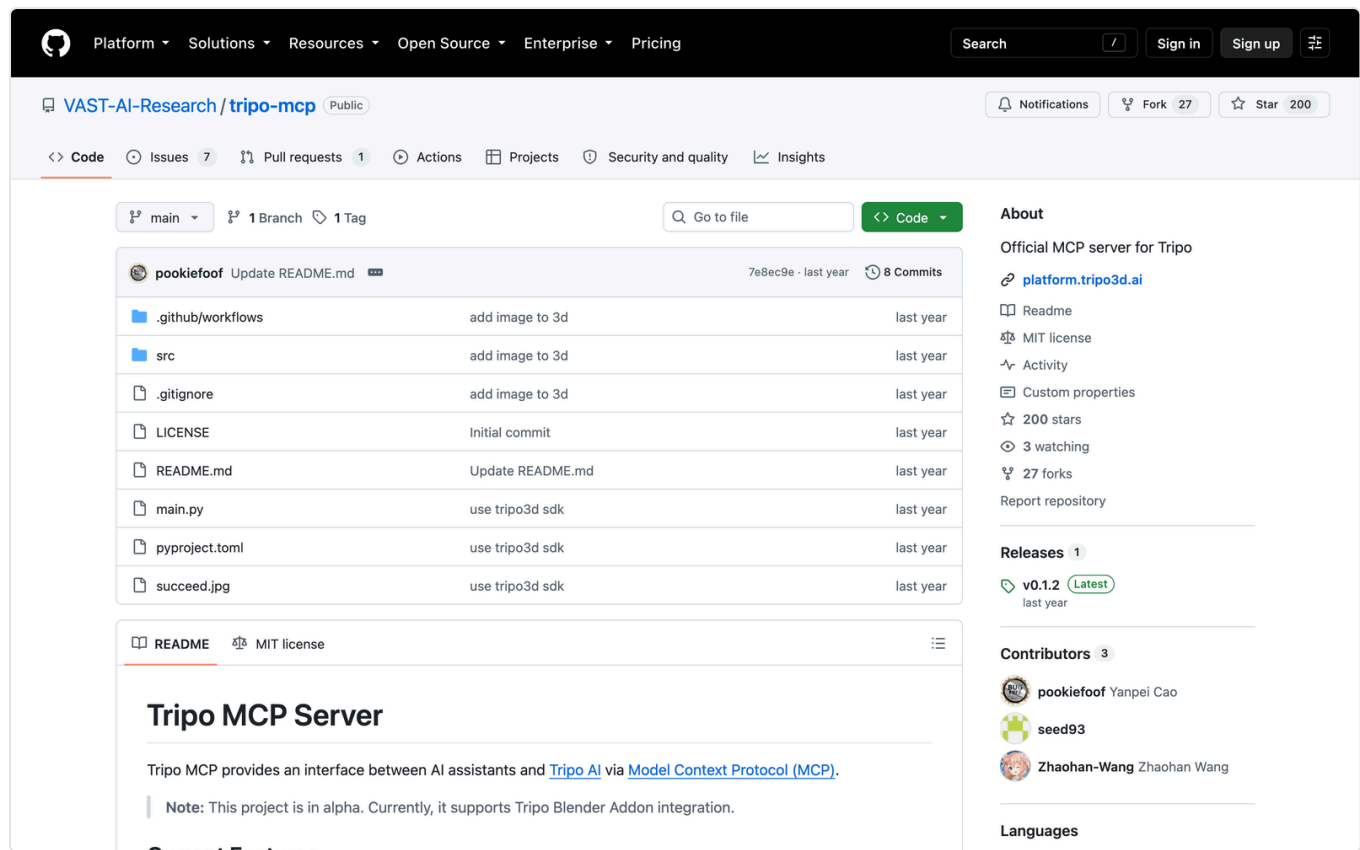
然后用 `claude mcp list` 验一下，我这台机器上打出来的是 `tripo: tripo mcp --quiet --no-open - ✓ Connected`。那两个参数是有理由的：`--quiet` 把进度日志从stderr上摁掉，免得和协议输出混在一起；`--no-open` 不让它自己弹浏览器——MCP是给Agent用的，不该在你没看着的时候开窗口。

注意

认证不写在这段配置里。server读的是 `tripo login` 写下的 `~/.tripo/config.json`，或者环境变量 `TRIP0_API_KEY`（前面引过的那句SKILL.md原文说的就是这两个来源）。所以先在终端 **`tripo login`** 一次，别把key抄进 `.mcp.json`——那个文件通常是要进git的。还有一条：项目级的 `.mcp.json` 第一次会停在 `Pending approval`，得在 `claude` 里批准一次才生效，这是客户端的安全设计，不是你配错了。

到这儿为止都是实测的。再往下——在会话里说一句「给我做个宝箱放到assets目录」——我没跑通，因为那句话最终会走到 `tripo_make`，而钱包是0。请求本身没有格式要求，SKILL.md给的示范就是这个句式：说清楚要什么、放哪儿，参数它自己填。

第二条是GitHub上的 `VAST-AI-Research/tripo-mcp` 仓库。采集当天我数了一下：200 star、27 fork、总共8次提交，最后一次提交停在2025年4月14日，到现在快十七个月没动过，7个issue开着，README自标alpha。



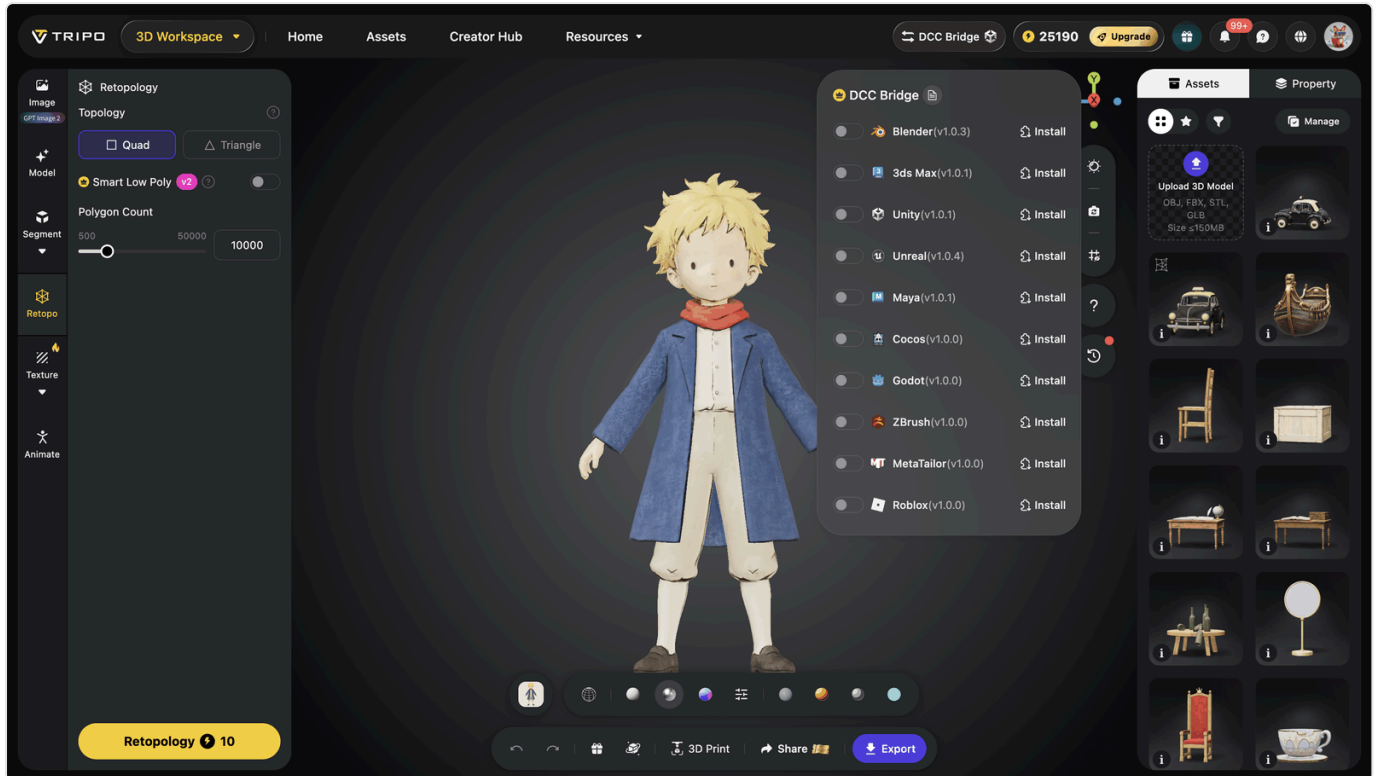
github.com/VAST-AI-Research/tripo-mcp, 2026-09-11采集。最新release是v0.1.2 (2025-04-12)。官方还有一篇博客教Cursor配uvx tripo-mcp 走的也是这个仓库，而那篇文章里「switch the Agent to claude-3.7-sonnet」这句话把它的年代钉死了，只能当历史路径读。

核心建议

要给Agent装Tripo的MCP，我会选CLI自带的 `tripo mcp`：它跟着CLI一起更新，不存在仓库停更的问题，工具表和调用通路我这次真跑过，配进Claude Code也确认过它认得。还没测到的只剩 `tripo_make` 本身——调它就要真扣积分，卡点和CLI那条是同一个。

DCC Bridge：唯一不涉及AI Agent的那条

这条很容易和MCP混淆，其实完全是两回事。**DCC Bridge**是一条传输通道：你在浏览器里的Tripo Studio生成好模型，点一下，模型直接出现在Blender里，不用下载、不用导入。它不替你调生成，也不听人话。



Studio顶栏点DCC Bridge弹出的面板，2026-09-11实测截图。十个目标软件各自带版本号：Blender(v1.0.3)、3ds Max(v1.0.1)、Unity(v1.0.1)、Unreal(v1.0.4)、Maya(v1.0.1)、Cocos、Godot、ZBrush、MetaTailor、Roblox（后五个都是v1.0.0）。定价页的文案只列了八个，面板里比文案多出MetaTailor和Roblox两个。

官方博客写的Blender版要求：Windows或Mac；Blender 4.1.0或更新；浏览器只支持Chrome 116+、Edge 116+、Opera 102+，其他浏览器不支持。装法是在Studio里下ZIP包，Blender里 编辑 → 偏好设置 → 插件 → 从磁盘安装 选那个ZIP（不要解压），启用后按 **N** 调出面板，回网页点DCC Bridge激活，然后在导出小窗里点 Export。

导出弹层那张图在 § 03。这里只看底下两个按钮：左边**Send To**就是DCC Bridge的出口，右边Export是普通下载。

两条官方FAQ提前记住能省时间：不能同时推给多个Blender实例，连接只认第一个启动的那个，开多个会报「Port already in use」；插件更新后如果显示「All connections closed」连不上，重启Blender并重新激活，看到日志里出现「Server running on port XXXX」才算真连上。

注意

DCC Bridge要Pro及以上套餐，免费账号没有。这和 § 03讲的导出墙是同一堵墙：免费账号拿不到文件，自然也不可能一键推进Blender。

哪些步骤必须你自己做

这一轮跑下来，有五件事Agent不该替你做，或者做不了。

1 浏览器授权页上点Authorize

设备码流程的全部安全性就在「人眼比对终端码和网页码」这一下上。这一步被代做，等于把这道闸拆了。

2 充值，以及领不领那个14天免费钱包

一个花钱，一个是一次性且带倒计时的权益。

3 保证浏览器里已经登录developers.tripo3d.com

没登录态的话授权页会先拦一道登录或注册，那一步可能带验证码或第三方登录。

4 额度耗尽之后怎么办

等重置还是买credits，同样是花钱决策。

5 `sudo npm install -g` 要的机器密码

本机不需要，但Node装在系统目录的机器会撞上。

最小项目长什么样

走CLI的话，最小项目就是一个空文件夹加一条命令：

```
my-game/  
├── tripo-out/  
│   └── <每次任务一个目录>/  
│       ├── model.glb  
│       ├── preview.png  
│       └── task.json  
└── index.html
```

`task.json` 记录这个任务是怎么创建的。官方给Agent的建议里有一条我很认同：如果 `preview.png` 存在，在决定继续管线还是重跑之前先看一眼它。这是让Agent自己做质检，比跑完一整条链再发现形不对便宜得多。

这一节还缺什么

缺的是最后一步：钱包里有积分之后，把 `tripo make` 真的跑通一次。具体缺的是三个数——任务从提交到成功的算子耗时、扣掉的API积分、以及 `tripo-out` 里那一个目录的真实产物。现在这一节能告诉你的是「到发起生成为止的每一步长什么样、会卡在哪」，还不能告诉你「一件资产从命令行出来要多久、多少钱」。CLI、Codex 插件、MCP三条路卡在同一个地方，就是那个0积分的API钱包。

核心建议

换个角度：缺的这三个数你自己能补，而且这一节已经把补它需要的东西全给了。命令是完整的那一条（`tripo make prop-02-bench.png --model tripo-p2 -p quad=true --no-open`），产物目录长什么样在上一小节，判「有没有白花钱」的退出码表在前面，`tripo balance` 是跑之前跑之后各敲一次、差值就是这一件的真实价钱。钱包一充上，从这一节最后一步到那三个数之间，只隔一条命令。

这一轮往前挪了一格：MCP那条补了协议实测和工具表，也把 `tripo mcp` 配进Claude Code验到 `Connected`，配法在前面那一小节。所以缺口从「装到哪一步会卡」窄成了一件具体的事——那一件落盘的资产。API那条的缺口更大，具体缺哪三样，在前面API那一小节已经点过名了。

那个「Get Your free wallet(14 days)」按钮一直在旁边，我没点，理由在前面那张钱包截图底下：它是一次性权益、带14天倒计时，该留给真正要连着跑测试的那一周，而不是为了在书里凑一个数把它烧掉。这一节宁可缺一个数，也不想给你一个「我领了、跑了一件、然后钱包过期」的样本。

§12 Three.js：让模型在网页里跑起来

Three.js — From GLB Files to a Playable Page

六个可交互demo跑下来，我在加载Tripo资产这件事上踩过的坑、一个和直觉完全相反的性能结论，以及P2.0那条路上完全不同的另一套工序。

先说三件决定成败的事

把Tripo导出的GLB塞进Three.js，理论上是一行代码的事。实际上我做完巴黎城、星月夜、四冲程发动机、旋转游乐场、游乐园、小王子星球这六个demo之后，发现真正决定它成不成的只有三件事。

一，环境贴图。没有它，Tripo出的PBR材质在白底上会渲成一团死黑，而你会以为是模型坏了。

二，贴图尺寸。卡顿的第一反应是三角形太多，但我量完之后发现渲染链只用掉了预算的八分之一，真正吃掉5 GiB显存的是贴图。

三，带骨骼的模型怎么复制。用错一个函数，一整排行人会像仪仗队一样同手同脚。

剩下的都是细节。这一节按这三件事展开，顺带把自校验那几个最贵的教训也放进来。



《巴黎来信》，Three.js加23件Tripo资产的可驾驶demo。建筑、载具、行人、地标是Tripo生成，街廓和河道是程序化。2026-09-10细节扩展版。

最小可跑的那个壳

开头那句「三行代码」在这一小节里兑现，同时也会看清它的水分：真正属于「加载模型」的只有 `new GLTFLoader().load()` 那一句，其余三十几行是把浏览器里那套渲染环境立起来的固定开销——渲染器、场景、相机、灯、环境贴图、一个能拖着转的controls。这一整页只写一次，后面六个demo都是在它上面长出来的。

先把环境立住。我所有demo的Three.js都是固定在项目vendor目录里的，不走CDN，因为发布包要能在断网环境下跑通、也要能塞进B站Toy这种子路径托管。

vendor目录里我只精选了五个文件：

```
vendor/
├── three.module.js
└── addons/
    ├── loaders/GLTFLoader.js
    ├── controls/OrbitControls.js
    ├── utils/BufferGeometryUtils.js
    └── utils/SkeletonUtils.js
```

这五个文件不用去页面上一个个找。npm 拉一次锁死版本的three，再从包里挑出来就行——我比对过，这么拿到的五个文件和我六个demo里实际跑的那五个逐字节相同：

```
npm i three@0.160.0
mkdir -p vendor/addons/loaders vendor/addons/controls vendor/addons/utils
cp node_modules/three/build/three.module.js vendor/
cp node_modules/three/examples/jsm/loaders/GLTFLoader.js vendor/addons/loaders/
cp node_modules/three/examples/jsm/controls/OrbitControls.js vendor/addons/controls/
cp node_modules/three/examples/jsm/utils/BufferGeometryUtils.js vendor/addons/utils/
cp node_modules/three/examples/jsm/utils/SkeletonUtils.js vendor/addons/utils/
```

版本号要钉死，不能写 `npm i three`。addons里的文件会跟着主库一起演进，五个文件必须来自同一个版本，混版的报错长得和缺文件一模一样。本节所有代码都在r160（也就是 `three@0.160.0`）上跑过。

核心建议

BufferGeometryUtils.js 不是可选的，哪怕你的代码一次都没import它。r160的 `GLTFLoader.js` 自己会import它，缺这一个文件，页面的表现是控制台一条404、画布压根没被创建、屏幕全白——看起来像模型加载失败，其实卡在模块解析。我在2026-09-13重跑下面这个壳的时候又撞了一次，所以写在这儿。

然后是壳本身。下面这一整页可以直接复制、另存成 `index.html`，和上面那个vendor目录、一个 `model.glb` 放在同一层：

```

<!doctype html><meta charset="utf-8">
<style>html,body{margin:0;height:100%;overflow:hidden}canvas{display:block}</style>
<script type="importmap">{"imports":{
  "three": "./vendor/three.module.js",
  "three/addons/": "./vendor/addons/"
}}</script>
<script type="module">
import * as THREE      from 'three';
import {GLTFLoader}    from 'three/addons/loaders/GLTFLoader.js';
import {OrbitControls} from 'three/addons/controls/OrbitControls.js';

const renderer = new THREE.WebGLRenderer({antialias:true});
renderer.setPixelRatio(Math.min(devicePixelRatio, 2));
renderer.setSize(innerWidth, innerHeight);
document.body.appendChild(renderer.domElement);

const scene = new THREE.Scene();
scene.background = new THREE.Color(0xf2f0ea);
scene.environment = studioEnv(renderer); // 没有它，金属件在白底上是死黑（函数写在本页末尾）
const camera = new THREE.PerspectiveCamera(45, innerWidth/innerHeight, 0.01, 100);
camera.position.set(2.2, 1.6, 2.8);
const controls = new OrbitControls(camera, renderer.domElement);
controls.target.set(0, 0.7, 0);
const sun = new THREE.DirectionalLight(0xffffffff, 2.5); sun.position.set(3, 5, 2);
scene.add(sun, new THREE.AmbientLight(0xffffffff, 0.5));

new GLTFLoader().load('./model.glb', g => {
  const root = g.scene, b = new THREE.Box3().setFromObject(root);
  const c = b.getCenter(new THREE.Vector3());
  const s = 1.6 / Math.max(...b.getSize(new THREE.Vector3()).toArray()); // 原始只有1.0单位
  root.scale.setScalar(s);
  root.position.set(-c.x*s, -b.min.y*s, -c.z*s);
  scene.add(root);
});
addEventListener('resize', () => {
  camera.aspect = innerWidth/innerHeight; camera.updateProjectionMatrix();
  renderer.setSize(innerWidth, innerHeight);
});
renderer.setAnimationLoop(() => { controls.update(); renderer.render(scene, camera); });

// 环境贴图: canvas 画一张渐变当 equirect, 不需要外部 HDR 文件。为什么这么写见下一小节
function studioEnv(rd){
  const c=document.createElement('canvas'); c.width=128; c.height=128;
  const g=c.getContext('2d'), grd=g.createLinearGradient(0,0,0,128);
  grd.addColorStop(0, 'ffffff');
  grd.addColorStop(.34, '#e8e4da');
  grd.addColorStop(.52, '#928d82');
  grd.addColorStop(.75, '#5c574e');
  grd.addColorStop(1, '#403c35');
  g.fillStyle=grd; g.fillRect(0,0,128,128);

```

```

g.fillStyle='rgba(255,255,255,.9)';
g.beginPath(); g.ellipse(40,24,26,13,0,0,Math.PI*2); g.fill();
const t=new THREE.CanvasTexture(c);
t.mapping=THREE.EquirectangularReflectionMapping;
const pm=new THREE.PMREMGenerator(rd);
const env=pm.fromEquirectangular(t).texture;
pm.dispose(); t.dispose(); return env;
}
</script>

```

在这个目录里起服务、浏览器开 `http://localhost:8765`，能看见模型、能拖着转，再往下读那三个坑：

```
python3 -m http.server 8765
```



上面那一页原样跑出来的样子，喂的是游乐园那座旋转木马的GLB。2026-09-13用无头Chrome (960×640) 实拍：顶棚、马匹、底座的金属描边都在，说明 `studioEnv()` 接上了。这就是「屏幕上有东西了」那一级台阶，后面三个坑都是在这一页上打的补丁。

注意

WebGL页面必须走http，不能双击开。 `file://` 协议下GLB加载会被CORS挡住，上面那条 `http.server` 就是为这个。纯程序化、零外部资源的HTML不受这条限制，双击能开。我六个demo同时开着的时候，端口是8765到8794排下去的。

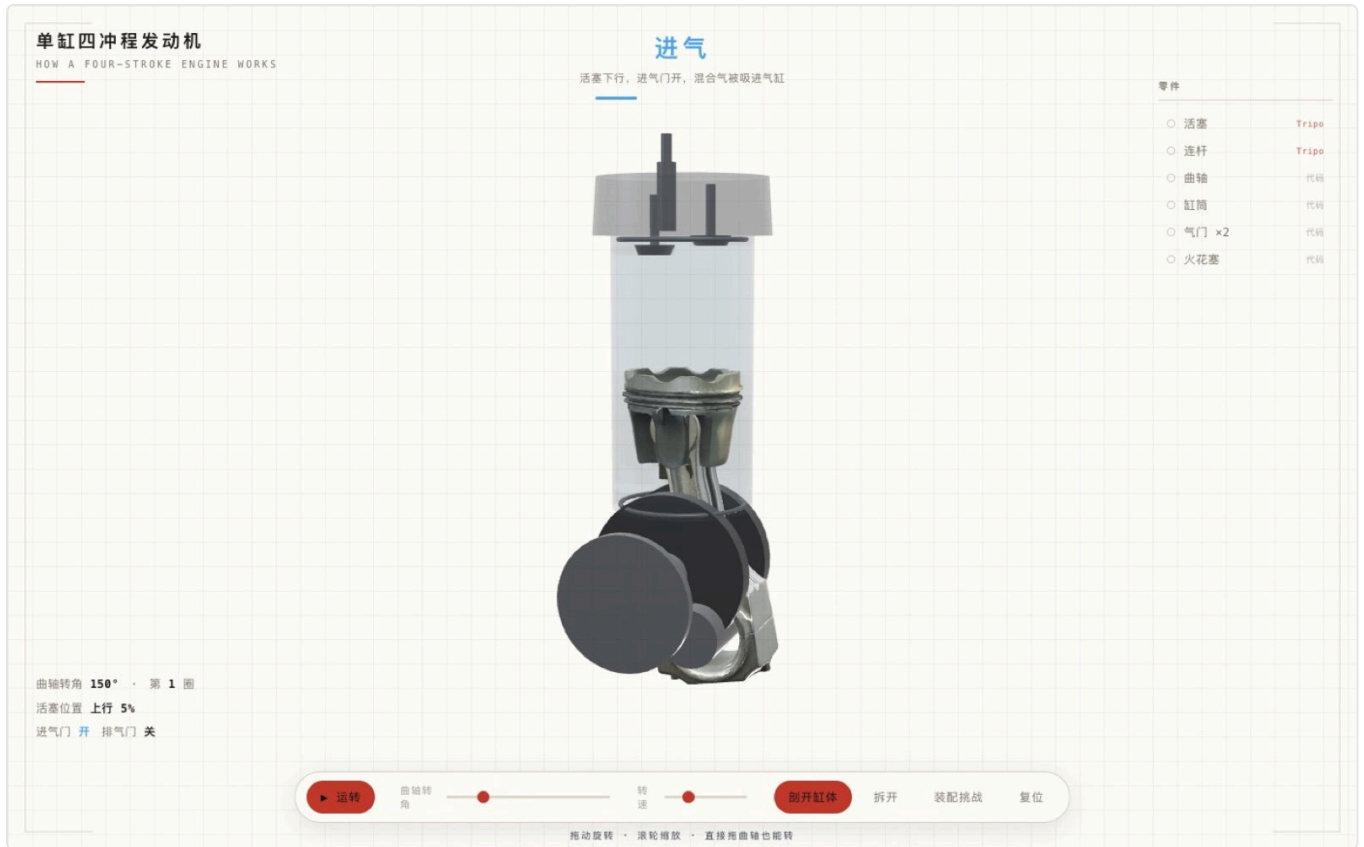
坑一：白底加PBR金属加没有环境贴图，等于死黑

这是我付出时间最多的一个坑，而且它有伪装。

四冲程发动机那个demo，v1是深色赛博底，看着一切正常。v2改成纸感白底之后，整台机器直接变成一团黑影。金属度高的时候漫反射被抑制，只靠直接光什么都看不见；深色背景下这个问题被背景吃掉了，一换白底就全暴露。



发动机demo v1，深色赛博底。金属件在深色背景下看着一切正常，问题被背景吃掉了。右边那栏零件表顺便标了每一件的来源：活塞和连杆是Tripo，曲轴、缸筒、气门、火花塞是代码，判据见 § 15。2026-09-10。



v2改成纸感白底。这一版是补上canvas合成的环境贴图之后的样子，金属件有了明暗对比；不补的话整台机器是一团黑影。2026-09-10夜。

解法不需要外部HDR文件，就是上面那一页末尾的 `studioEnv()`：用canvas画一张渐变当equirect环境贴图，走 `PMREMGGenerator` 转成可用的环境光照，再挂到 `scene.environment` 上。

那十几行里有两个细节是试出来的，不写出来你照抄也调不对。

环境贴图不能全亮。我第一版渐变是从白到浅灰，结果金属反射回来全是亮色，整个画面发白、没有重量感。必须给它一个暗部，相当于摄影棚里的黑旗，金属才有明暗对比。上面那段渐变从 `#ffffff` 压到 `#403c35` 就是为了这个。

白底场景别用ACESFilmic色调映射，它会把白底压成灰。

星月夜那个demo走的是另一条更简单的路：加载完直接把材质的金属度归零。

```
// 星月夜：村庄模型本来是一团黑，两行就回来了
mat.metalness = 0;
mat.roughness = 0.92;
```

两条路的取舍很清楚：要金属质感就老老实实做环境贴图，不要金属质感就把`metalness`摺掉。最怕的是既不做环境贴图又保留高金属度，那就是一团黑。

坑二：尺寸和原点，两件事都不能想当然

Tripo导出的GLB被归一化到最长边约等于1.0单位（不是高度：解析包围盒能看到长椅是 $1.0 \times 0.549 \times 0.413$ 、出租车是 $0.522 \times 0.570 \times 1.0$ ，撑满1.0的那条边各不相同）。我在 § 14 用实测数字展开这件事，这里只说Three.js侧怎么处理：实例化之后先量包围盒，再按目标尺寸算缩放，最后把底面压到地面。

```
const b = new THREE.Box3().setFromObject(root);
const size = b.getSize(new THREE.Vector3());
const center = b.getCenter(new THREE.Vector3());
const scaled = new THREE.Group();
scaled.scale.setScalar(height / size.y);
root.position.sub(new THREE.Vector3(center.x, b.min.y, center.z));
scaled.add(root);
```

这段对静态模型是对的。对带骨骼的模型是错的。

`Box3.setFromObject` 量 `SkinnedMesh` 量不准。（我一开始以为是「它拿的是绑定姿势的几何包围盒」，后来才弄清楚：three.js从r155起已经会按当前姿势算了，真正咬人的是那个包围盒缓存不会自己更新——姿势变了，你量到的还是上一帧的。修法一样，理由不一样，这里更正一下。）叠上Armature节点变换之后算出来的尺寸离谱。做方案D那个「同一模型两种绑定」的实时对照页时，右边那个模型被放大到画面里只剩两只鞋。

解法是绕开Three.js，直接读GLB里 `POSITION` accessor的 `min` 和 `max` 拿真实尺寸，按已知值硬算。GLB的头部结构很简单，不需要任何库：

```
import struct, json
d = open(f, 'rb').read()
jl = struct.unpack('<I', d[12:16])[0]
j = json.loads(d[20:20+jl])
# 遍历 meshes[].primitives[].attributes.POSITION → accessors[i].min / .max
```

顺带一条很容易误判的：同一批资产的原点可能不一致。 实测Blender导出的那版模型脚在 `y=0`，Tripo导出的那版中心在 `y=0`，而两者几何高度都是1.0。不知道这件事，你会以为是缩放算错了，然后一路去查缩放。

坑三：带骨骼的模型不能用普通clone

巴黎城里有18个行人，其中6个用的是同一个绑骨模型（另外12个是静态的，为什么不是18个全走，见 § 18）。第一版我用 `clone()` 复制，结果这几个人同手同脚，像在走正步。

原因是普通clone会让所有实例共享同一副骨骼。必须用 `SkeletonUtils.clone()`，然后每个实例配一个自己的 `AnimationMixer`。

```
import {clone as skeletonClone} from 'three/addons/utils/SkeletonUtils.js';

const root = skeletonClone(walkScene);
const mixer = new THREE.AnimationMixer(root);
mixer.clipAction(walkClip).play();
// 相位错开, 否则一整排人还是整齐划一
mixer.setTime((x*0.37 + z*0.11) % walkClip.duration);
```



同一个行人, T1和相隔0.7秒的T2。姿态和位置都变了, 说明骨骼动画真的在跑, 不是模型在平移。这是巴黎demo的自检截图, 2026-09-10。

注意

SkeletonUtils.js 不在精简版vendor目录里。我的vendor原本只放了GLTFLoader、OrbitControls、BufferGeometryUtils三个, import它会让整个页面启动失败, 而且报错信息指向的是模块解析、不是缺文件。补法就是本节开头那串 cp 的最后一行, 版本必须和另外四个一致 (我用的是r160)。

还有两条是接Tripo绑骨动画特有的。

把位移轨切掉, 只留旋转轨。retarget出来的剪辑里 Hip.position 的Y位移范围是1.338, 而人本身才1米高。行走是代码推的, 动画再叠一层位移, 循环回到起点时整个人会弹回去。我在画面里看到的「走着走着又会后退一大步」, 根因就是这个。步态本来就是纯旋转, 位移归游戏代码管。

在Tripo那边对应的是导出时勾上「原地播放动画」(见 § 06), 勾了它动画就只管迈腿。

模型朝向是试出来的, 不是推出来的。Tripo预设人形的自然朝向不是 +z, 实测差90度。小王子那个demo里这个常量写死成 PRINCE_YAW = -Math.PI/2, 而这个数是在浏览器控制台里现场转 rotation.y、看正对还是背

对镜头试出来的。

但「差90度」这句话只对绑骨预设人形成立，别把它当成通则。静态资产还有另外两套完全不同的惯例，符号甚至是相反的，我在后面「朝向」那一节按类别拆开写。把三类混成一句话记，迟早会把一栋楼装反。

性能：瓶颈是贴图，不是三角形

这是这一节我最想让你记住的一条，因为它和直觉完全相反。

小王子星球那个demo做完，页面卡得没法看，我丢给agent的原话是「性能需要优化500%」。我的第一反应和所有人一样：减面、关泛光、砍阴影。

但我先量了一次。量出来的结论把我的第一反应全推翻了。

指标	实测值（M4 Pro / 3024×1722画布）
整条后期链GPU耗时	0.37 ms（场景+阴影+泛光+调色，520万像素）
主线程JS	1.5 ms/帧
三角面/绘制调用	147k-285k / 25-56
60 fps的预算	16.7 ms

渲染链只用掉了预算的约八分之一。卡顿不可能来自三角形数量、后期链或者JS，砍这些等于白砍。

真凶在显存。把场景里所有材质的贴图列出来：

唯一贴图数：31

尺寸分布： 8192×8192 × 10 4096×4096 × 20 2048×1024 × 1

估算显存： 5.0 GB (含 mipmap)

这批v3.1资产导出的是8192²的PBR贴图。（贴图这件事必须按模式说：v3.1这条路出base color、normal、ORM三张，P2.0那条路只出一张base color，两者的处理方式不一样，后面有一整节讲P2.0。）每个道具在画面里才一两米高、屏幕上几百像素，却挂着一张8192²的底图。一张8192²按RGBA32展开是268 MB，加上整条mipmap链（×4/3）约358 MB；17个资产的贴图加起来是5.0 GiB，GPU一直在做纹理换入换出，表现出来就是「卡」。（别拿17乘358——那17件挂的不全是8K：唯一贴图31张，其中8192²的10张、4096²的20张、2048×1024的1张，10×358+20×89.5+11=5381 MB=5.01 GiB，逐张清单在 § 22。）

核心建议

先把这个数的口径钉住，因为它在书里被引了好几次。上面那段是脚本原样的输出，它按1024进位算完、却把单位打成了「GB」——那一行的正确写法是**5.0 GiB**。同一批贴图按每像素4字节展开、再乘整条mipmap链，用本书算显存的十进制口径（1 MB=100万字节）严格算下来是**5.38 GB**：两边差约7%，是同一笔账的两种进位，不是两批贴图。**显存这笔账，本书的规矩是量出来的读数写GiB、自己算出来的写GB**，看单位就知道这个数是量的还是算的。下面那张优化前后对照表里的两个数出自同一次测量、同一个进位（5.0 GiB和0.22 GiB，折成十进制是5.38 GB和0.24 GB），所以**倍数不受口径影响**，23×在哪一边都成立。

处理办法是只降贴图，不碰几何：

```
gltf-transform resize 产出/x.glb 网页/assets/tripo/x.glb --width 1024 --height 1024
```

（gltf-transform 是装成全局或本地依赖之后的简写。没装的话，在前面补上 `npx @gltf-transform/cli`，效果一样，只是每次多等几秒启动；说明见 § 04。这条命令要先有Node，见 § 11。）

核心建议

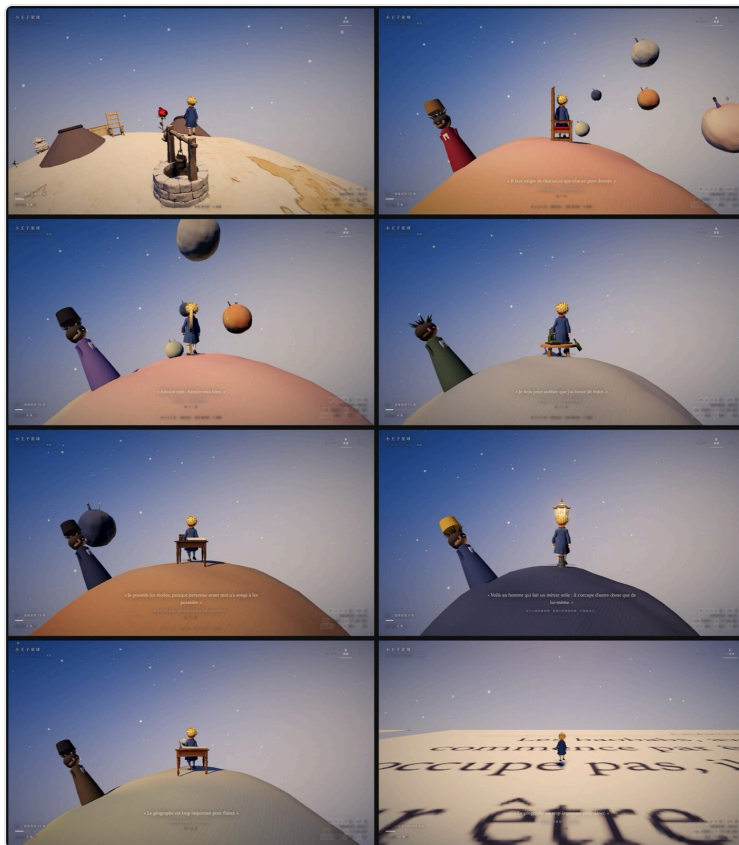
用 **resize** 而不是 **optimize**。optimize 默认会顺手跑 simplify 减面，而几何根本不是瓶颈，没有理由动它。主角是特写对象给2048²，其余1024²。

	改前	改后	倍数
贴图显存	5.0 GiB	0.22 GiB	23×
GLB载荷	89.87 MiB	12.30 MiB	7.31×
资产就位	3.90 s	0.70 s	5.6×
JS每帧（中位/p90/最大）	1.5 / 2.9 / 4.4 ms	0.7 / 0.8 / 1.2 ms	2.1×
三角面	147,879	147,879	未动

注意

GLB载荷那一行我重量过，和当时的记录对不上一个数，照例摆在这儿。当天记录里这一行写的是「97.6 MB → 12.3 MB，7.3倍」。倍数是对的：我把部署出去的那17件重新量了一遍，优化后合计**12.30 MiB**，对应的源文件合计**89.87 MiB**， $89.87 \div 12.30 = 7.31$ 倍，和当时记的7.3倍吻合，所以12.3那个数是1024进位的读数。**对不上的是97.6**——这个数我没能复现出来，产出目录里那19个GLB加起来是94.76 MiB（99.36 MB），都不等于它，很可能它统计的范围里含了没部署的文件。表里现在填的是我能复现的那一组。

画质在这个观看距离上肉眼无损。王座特写的木纹、红绒、金冠，主角特写的脸、外套扣子、围巾，1024²都撑得住。



小王子星球的道具接入自检，八格一次看完七颗星球加一格C观。贴图全部降到1024²之后截的，王座、镜子、酒瓶桌、书桌这些特写对象的材质细节都还在。2026-09-11。

顺带省掉的四样，都是量完之后才敢动的。

antialias 关掉。接了 EffectComposer 之后画面是先渲进render target的，画布那层的MSAA对场景几何完全不起作用，开着只是白付一次多重采样缓冲加resolve。

阴影从 PCFS_{Soft}ShadowMap 降到 PCFShadowMap，并且不再每帧重渲阴影贴图。

泛光跑在半分辨率上。纯填充操作，肉眼分不出，模糊开销少四分之三。

自适应分辨率兜底，但中间要留死区。帧时间掉到48 fps以下降档，回到91 fps以上才升回去。两边挨太近的话，一台稳定跑50 fps的机器会来回抖，而每次改档都要重建WebGL缓冲区，那比稳定50 fps更难受。



小王子星球A观「原作星球」，B612直径28米。贴图优化之后锁定60 fps。星球几何是代码算的（球面数学自己算更准），表面那张等距圆柱贴图是Codex出的。2026-09-10。

另一条路：P2.0资产的接入工序链

上面这些坑，全是v3.1那条路上踩出来的。九月我把巴黎那座城里沿街资产整批换成P2.0重出的件，才发现这是另一条路，坑几乎不重合。同样是「把GLB塞进Three.js」，工序链要重走一遍。

五道工序，少一道就装不上去。

1 转格式

P2.0带贴图的产物只给FBX，浏览器不认，先转GLB。这一道的做法和怎么证明它没弄坏东西，在 § 13。

2 降8K贴图

不降就是在纸上就能算出来的显存灾难，数字在下面。

3 定yaw

P2.0的自然朝向和v3.1的旧件差90度，而且每一件都得自己定。

4 烘变换

把旋转和缩放乘进顶点，不要写成节点矩阵。理由在你自己的加载代码里。

5 量A/B帧时间

换完拿旧件的读数对一遍，确认没把性能换坏。

核心建议

耗时口径先说清楚。这一节里所有带秒数的都是本机跑脚本的墙钟。书里凡是写Tripo侧生成、拆件、绑骨的秒数，一律是算子耗时口径，不含上传与排队，端到端会比它长一截。两边不能直接相加。

第一件事：P2.0只出一张base color

这条我放在最前面，因为它决定了你到底在找什么文件。

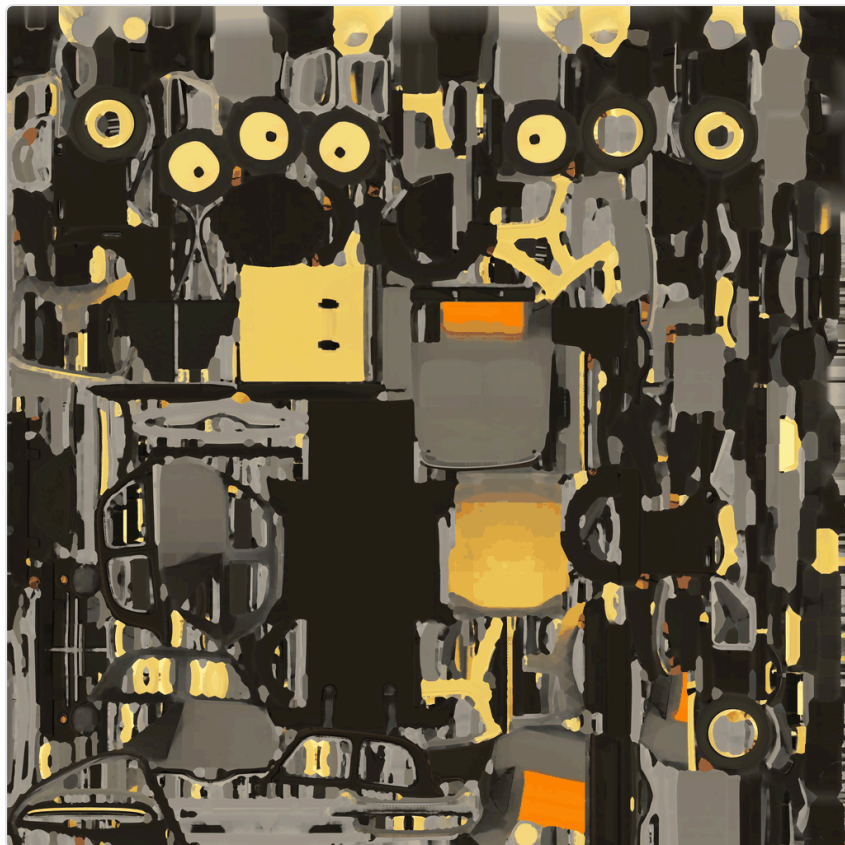
模式	出几张贴图	材质参数
v3.1（高精度模型）	base color、normal、ORM三张	metallic=1 / roughness=1，真正的粗糙度和金属度由ORM那张图逐像素给
P2.0（智能网格）	只有base color一张	转出来是 metallic=0 / roughness=0.9，全模型一个值

ORM是把环境光遮蔽、粗糙度、金属度三个量打包进一张图RGB三个通道的常见做法，是引擎判断「这块表面是金属还是布」的依据。P2.0没有它，也没有法线图，于是同一盏灯下，P2.0的新件会比v3.1的旧件哑：没有法线图，表面那层细凹凸不参与打光；没有ORM，整件东西从头到脚只有一个粗糙度。

说清楚一点：P2.0这个模式本身就只出这一张图，转换过程什么都没丢。我用两份独立记录核过：一份是taxi那次FBX转GLB的逐项核验，一份是38件批量降贴图的逐件日志，两边都只数到一张。

注意

所以别拿着「Tripo出PBR三件套」这句话去P2.0的产物里找法线图，那个文件不存在。怎么避开：走这一节前面那两条路之一，要么老老实实做环境贴图把高光和暗部撑起来，要么干脆把 metalness 搥到0、roughness 给足，让它以哑光材质的身份好看。哑光是个正经外观，发黑才是坏的。真要凹凸感，就去Blender那一侧自己烘一张法线图（见 § 13）。



P2.0那辆出租车的贴图，降到 1024^2 之后存出来肉眼检查的那一张。这就是全部：车身、轮胎、车灯、顶灯的颜色都摊在这一张图集上，没有配套的法线图，也没有ORM。原始尺寸 8192^2 、6.81 MiB，降完308 KiB。2026-09-11。

第二件事： 8192^2 乘26件等于9.3 GB

P2.0那张base color出厂就是 8192^2 。前面小王子那节已经量过一次8K贴图的代价，巴黎这边我又算了一遍，因为件数不一样（这座城里装机的是26件，重出的那批一共38件），而且这笔账在纸上就能算完，不用等浏览器崩给你看。

注意

这个26不是§16里那个25，两个数各有各的口径，别放在一起减。26数的是巴黎demo的 assets/ 目录里实际躺着的GLB文件个数——显存是按文件算的，一个文件一份贴图，所以下面这笔账的乘数只能是它，不能是人眼看得见的件数。（这笔账本身是个假设：如果这26个槽位全换成P2.0的 8192^2 原始件，要多少显存。）25数的是玩家在城里能看见的Tripo网格。差的那一个是 pedestrian-01：静态版和绑骨行走版是两个文件、同一个网格，显存要算两份，人眼只看得见一个人。§16那张「23、25、29数的不是同一批东西」的对账表把这一族数全列齐了，要对账去那儿。

贴图配置	每件显存	26件合计
P2.0原始 8192^2	358 MB	9.3 GB
降到 1024^2	5.59 MB	145 MB
v3.1旧件 512^2 三张	4.19 MB	109 MB

算法是按每像素4字节（也就是RGBA32）解码再加整条mipmap链，系数约1.33倍，和前面小王子那节写的358 MB是同一笔账。这笔账我按1 MB等于100万字节算，你要是拿显卡工具对，它按1024进位、读数会比这里小约7%（同一个数在那边是341 MiB、8.67 GiB），不是算错了。降到1024²之后比旧件贵33%，换来的是单张贴图分辨率翻倍，落在预算内。

1024这一档是我照配方定的，没和512、2048做过A/B。画质上够用，但我不能说它是最优解。

38件批量降贴图：不走Blender，直接改glTF的JSON和BIN

这批资产已经是GLB了，所以这一道我刻意绕开了Blender，直接改GLB里的glTF JSON和BIN数据块：只动images那几段字节和可选的顶点数据，其余原样搬过去。绕开Blender的理由、源与产物的逐项对照、以及38件全过的八项核验清单，完整在 S 17 「批量那一档」，那里还多两条：doubleSided 全是true的Blender成因，以及多张贴图必须按原图名一一对应。

朝向：三类资产三套惯例，千万别并排读

「Tripo出来的东西朝哪边」这个问题没有单一答案。我手上这三类资产各有各的惯例，混成一句话记一定会出事，所以拆开写。

资产类别	自然朝向	接进代码时
Tripo预设人形（绑骨那条路）	和 +Z 差90度	小王子demo里写死 PRINCE_YAW = - Math.PI/2
v3.1静态资产 · 行人	+Z	换成P2.0件时 yaw = -90
v3.1静态资产 · 建筑	-Z（宽立面在±Z，窄面在±X）	换成P2.0件时 yaw = +90
P2.0家族	+X，38件里36件如此	两个例外见下

先看行人和建筑那两行：符号是相反的，而两者的包围盒长得一模一样，光看数字分不出来。我第一次就是拿「旧件朝+Z」这一句去套建筑，把楼装反了90度，素墙对着马路。

P2.0家族的两个例外也都是渲图读出来的。vehicle-04-citroen-ds 正面朝 -X：+X 那一视图是尾灯和后窗，-X 才是DS那张四头灯的脸。building-10-bouquiniste-wall 正面朝 -Z：-Z 视图里书箱是敞开的、看得见书，+Z 是合上的箱盖。

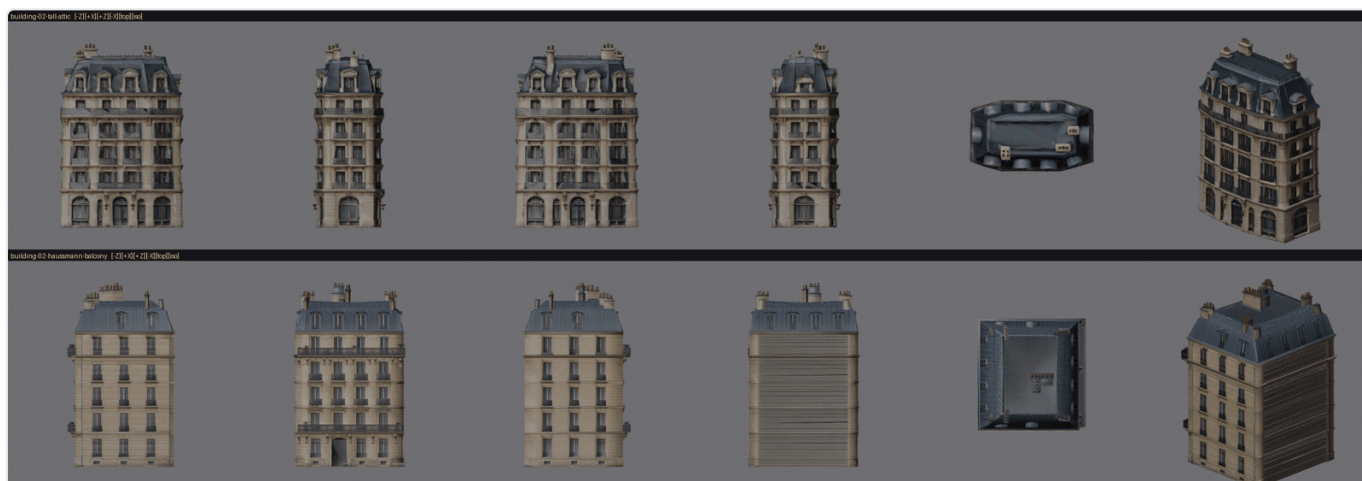
注意

四视图分不出正面和背面，必须渲六视图。常规四视图是 -Z / +X / +Y / 斜45度，缺的正是 +Z 和 -X 这两个机位，而正背面的差别恰好只在这两个机位上看得见。判据只有一句：在哪个视图里看见正面，正面就朝那个方向。这一条不受相机roll影响，也不需要读任何数字。

再强调一遍这件事为什么不能靠算：**包围盒推不出朝向**。绕Y转90度只是把X和Z互换，转180度连互换都没有。`--yaw 90` 和 `--yaw -90` 出来的包围盒完全相同，一个车头朝前，一个倒着开。新旧各渲一组六视图并排看，这是唯一靠得住的办法，26件就得看26遍，没有自动化的捷径。



六视图并排：上排是被替换的v3.1旧件 `pedestrian-02-woman-dress`，下排是P2.0新件 `person-03-office-woman-baguette`。列的顺序是 `-Z` / `+X` / `+Z` / `-X` / 顶视 / 斜45度。旧件的正脸出现在第三列（`+Z`），新件的正脸出现在第二列（`+X`），差90度，所以 `yaw = -90`。2026-09-12本地Blender渲染。



同一套机位下的建筑。上排旧件 `building-02-tall-attic` 宽立面落在 $\pm Z$ （第一、三列），窄面在 $\pm X$ ；下排新件 `building-02-haussmann-balcony` 的正立面在第二列（`+X`），第四列（`-X`）是那面没有窗、只有横向砖纹的背墙。建筑这一类要 `yaw = +90`，和行人反号，而这件事只有并排看六视图才看得出来。

烘进顶点，不要写节点矩阵

`yaw`定好了，接下来是把这个旋转写到哪儿去。GLB里有两个位置可以放：写成场景根节点的变换矩阵，或者直接乘进每一个顶点。这里必须选后者，而理由不在Tripo那边，在你自己的加载代码里。

巴黎demo的 `loadCityTree()` 是这么写的：用 `Box3.setFromObject` 量世界包围盒（这一步会把节点矩阵算进去），然后把归一化施加在原始geometry上（这一步无视节点矩阵）。量的时候算了，用的时候没算，两步不在同一个坐标系里。给这样的加载器一个带旋转的节点矩阵，模型会静默错位，不报任何错。

烘进顶点对两条路都安全，所以我每一件都带着 `--bake` 跑。「量的时候算了、用的时候没算」这类不对称，是接第三方资产时最容易中一类埋伏，症状永远长成同一个样子：位置不对，但控制台干干净净。

A/B帧时间：量出来的是「量不出回归」，不是「变快了」

换完六件之后要回答一个问题：新件有没有把性能换坏。问题看着简单，第一次我量错了。

「六件」也要说清口径：它是2026-09-12那一轮真正落地的六个旧槽位——那一轮清单上排了九个，三个当场否掉（映射表把一栋六层公寓写成了地铁口、新咖啡露台多一根两米高的取暖伞会把桌椅按比例压到半米、行道树在另一条线上换），理由在 S 16。这一轮之后另一条线又换掉两个行人槽位，加起来才是 S 16说的「8个槽位」。下面这张A/B表量的是那六件，不是八件。

不能看帧间隔。浏览器的帧间隔被vsync锁在8.3或者16.6毫秒上，两组资产量出来的中位数会一模一样，差别全被锁没了。要量的是每一帧GPU真正忙了多久，这需要一个WebGL2扩展挂在 `requestAnimationFrame` 上：

```
const ext = gl.getExtension('EXT_disjoint_timer_query_webgl2');
const q = gl.createQuery();
gl.beginQuery(ext.TIME_ELAPSED_EXT, q);    // 这一帧开画之前
// ... renderer.render(scene, camera) ...
gl.endQuery(ext.TIME_ELAPSED_EXT);         // 画完
// 隔几帧再来收结果，单位是纳秒；GPU_DISJOINT 为真说明这次计时不可信，丢掉
if (gl.getQueryParameter(q, gl.QUERY_RESULT_AVAILABLE)
    && !gl.getParameter(ext.GPU_DISJOINT_EXT))
    ns = gl.getQueryParameter(q, gl.QUERY_RESULT);
```

协议也要钉死，否则你量的是自己的驾驶技术：刷新页面、跑一段自由漫游、回到出生点静止采样。巴黎demo的出生点坐标是确定性的，两次完全可复现。

	旧6件 (v3.1)	新6件 (P2.0)
GPU中位	4.24 ms	3.87 ms
GPU p95	11.72 ms	5.40 ms
三角面	3,247,182	3,224,973
draw call	212	212
控制台报错	0	0
采样	151帧	302帧

这张表的结论是「量不出回归」，不是「变快了」。0.37毫秒的差落在跑间噪声里，我没有第二次独立跑来支撑「变快」这个说法，所以不说。draw call一个没变，三角面降了0.7%，和逐件的算术对得上，这两条才是真结论。把噪声读成改进，是性能测量里最常见的一种自欺，尤其在你正好希望它变快的时候。

注意

测性能必须串行，不能和任何别的渲染任务同时跑。有一轮我在另一条线上用headless Playwright录B-roll，两边抢同一块GPU，量出来是中位8.57毫秒、p95 61.8毫秒、最小值3.19毫秒，方差大得离谱。那个最小值和机器干净时量到的3.29毫秒吻合，所以我判断是机器被占用而不是性能回归。**但这是推断，不是测量。**这种数据唯一正确的处理是丢掉重测，而不是挑一个看着顺眼的统计量写进结论。

一个格式坑：normalized的int16会让你算出32783倍的缩放

这个坑值得单独写，因为它完全不像格式问题，倒像「我的数学写错了」。

巴黎demo现有的26件资产都带 KHR_mesh_quantization 扩展，顶点位置不是float32，而是 normalized: true 的int16。这个标志的意思是：这里存的整数不是坐标本身，读的时候要除以这个类型的满量程（int16是32767）才是真实值。它是压缩类扩展的标配，因为整数比浮点省一半空间。

我自己那个无依赖的GLB解析器一开始不认这个标志，把原始整数直接当世界坐标读。于是 pedestrian-02 被读成**32767单位高**，而它实际是1.0。接着「按旧件高度对齐新件」那个功能拿这个数去做除法，算出来的缩放系数是32783。

```
# 按分量类型归一化回 [-1,1] / [0,1]
if accessor.get('normalized'):
    SCALE = {5120:127.0, 5121:255.0, 5122:32767.0, 5123:65535.0}
    pos = pos / SCALE[accessor['componentType']]
    if accessor['componentType'] in (5120, 5122):    # 有符号的还要夹一下下界
        pos = np.maximum(pos, -1.0)
```

两条可以带走的。自己写glTF解析器，就一定要处理 **normalized**。而从FBX那条路转出来的GLB不产生 normalized accessor，所以同一个解析器读新件一直是对的、只在读旧件时出错。一个只在一半输入上出错的 **bug**，比全错的bug难找得多，因为你会先怀疑那一半输入。

后期链里最深的一个坑：ShaderPass会clone uniforms

这一个我排查了很久，因为症状极具误导性。

```
const Grade = { uniforms:{ uRes:{value:new THREE.Vector2(1,1)}, ... } };
const gradePass = new ShaderPass(Grade);           // ← 这里 clone 了一份
Grade.uniforms.uRes.value.set(w, h);               // ← 写的是原来那份，着色器读的是 clone
```

两个对象，写错了不报错，着色器只会一直用初始值。

于是 uRes 停在 (1,1)，湿边效果「取邻居」取的是屏幕另一头的深度，整片地面被判成巨大跳变、压成灰的；而天空因为处处深度都是1.0反而看着完全正常。顺着「天空没事、地面全黑」这条线去查，会一路怀疑深度纹理格式、线性化公式、掠射面梯度，全都对，因为根本不是那些。

排查路径记下来，下次照做：

1 做差

把开关前后的两张图做 difference 合成，立刻看出「暗在哪儿」（天空纯黑、地面全亮）。

2 把中间量画出来

给着色器加一个 `uDbg`，直接输出深度或边缘强度。

3 三通道一起输出

`vec4(原始深度, 线性化深度, 边缘强度, 1)`，一次对齐三者。

4 最后才回头查uniform传没传进去

而这一步本该在前两步之前。

同一轮里我还清掉了两个自测脚本自己的bug。测量方法本身也要被怀疑。

自校验：这一节的教训最贵

截图自校验之前，先确认页面的加载遮罩已经淡出。做发动机v2那轮，我有四轮都在给一张蒙了白色遮罩的图调材质。每轮都判断「还是发白」，于是继续调环境贴图、调曝光、调金属度，而真正的原因是 `#Load` 那层不透明白色盖在整个画面上没消失。

注意

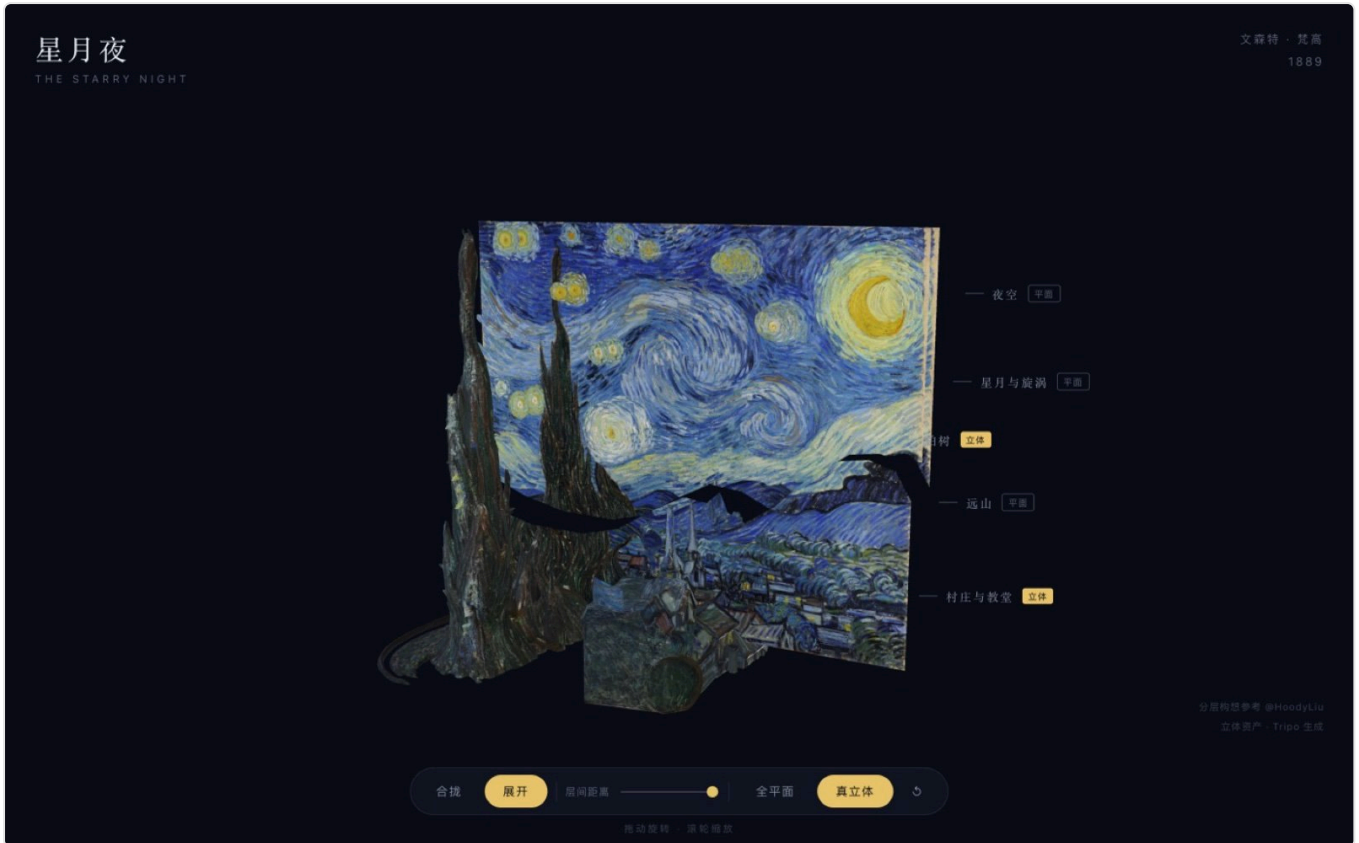
判据：截图里如果整体发白或发灰、且各元素均匀受影响，先查有没有覆盖层，别动材质。自检副本里直接把遮罩设成 `display:none` 最省事。

另外三条工程上的：

macOS没有 `timeout` 命令。`timeout 50 chrome ...` 会静默变成 `command not found`，整条命令什么都没干。我因此以为headless渲染失败了好几轮。

headless Chrome写完截图文件后不退出。命令会超时报错，但文件已经写出来了。报FAIL不等于没文件，收尾要按文件是否存在重新盘点，别信退出码。另外多个headless实例堆积会互相干扰，先 `pkill -f "headless=new"` 再跑。

后台标签页的 `requestAnimationFrame` 会停。本地自校验要加 `setInterval(frame, 340)` 兜底。



《星月夜》可展开立体油画，展开态。五层里柏树和村庄是Tripo生成的3D件，其余三层是带alpha的平面层。镜头必须侧过来，正视角下五层分开你看不出来。2026-09-10。

发布：两个沙盒的坑

做完了要给人看。静态托管最省事，把目录整个传上去就行，Vercel、Cloudflare Pages、GitHub Pages都能直接扔。

但有两类环境会咬人。

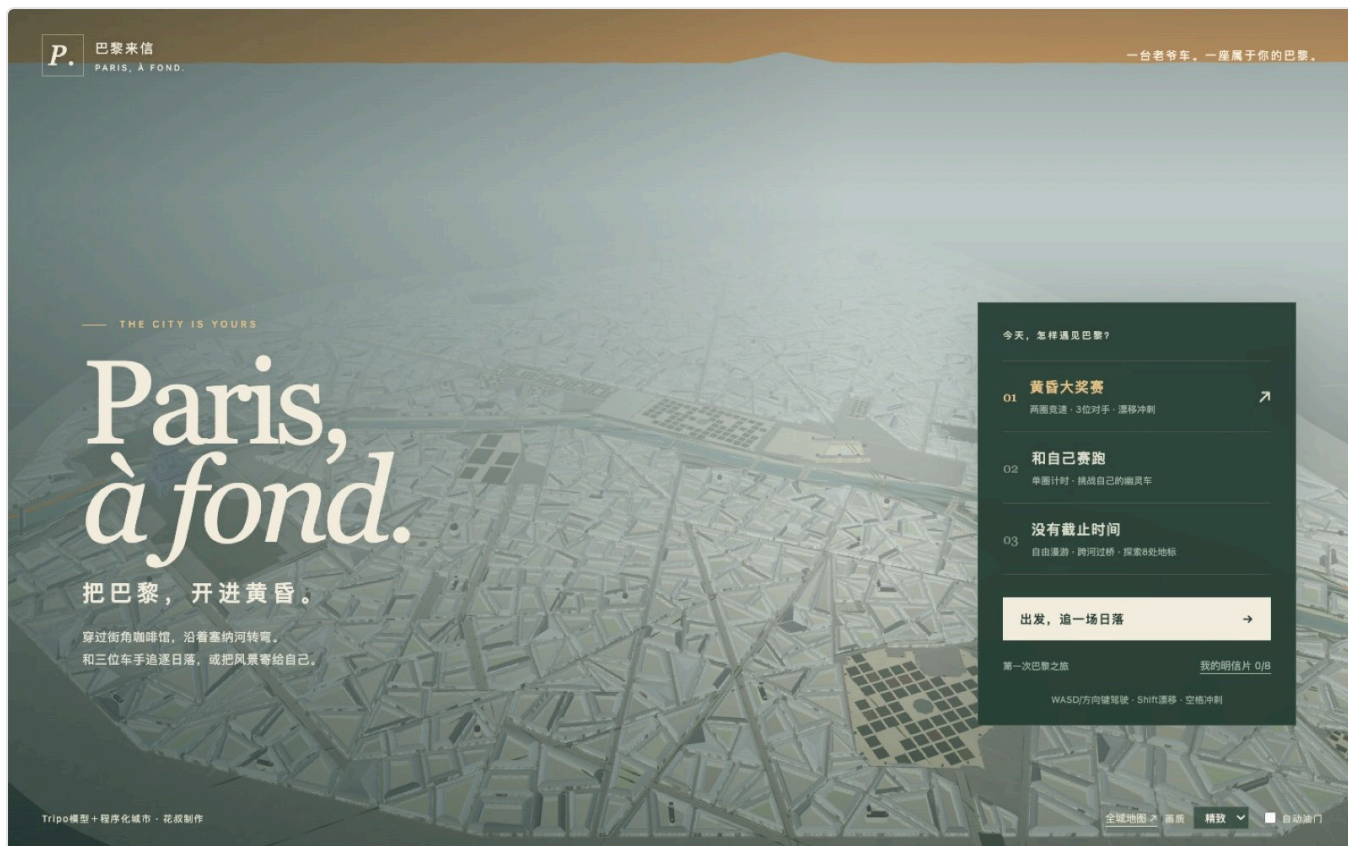
第一类是禁用 `fetch()` 的沙盒。我最早那版巴黎demo发在Artifact里，模型渲染成纯黑，没有任何报错。查了很久才明白：那个环境会完全禁用 `fetch()`（连 `data: URI` 都不放行），而Three.js的GLTFLoader加载贴图时优先走 `ImageBitmapLoader`、内部用的正是 `fetch()`，于是静默失效。

```
// 逼 Three.js 退回到走 <img> 标签的经典加载路径
window.createImageBitmap = undefined;
```

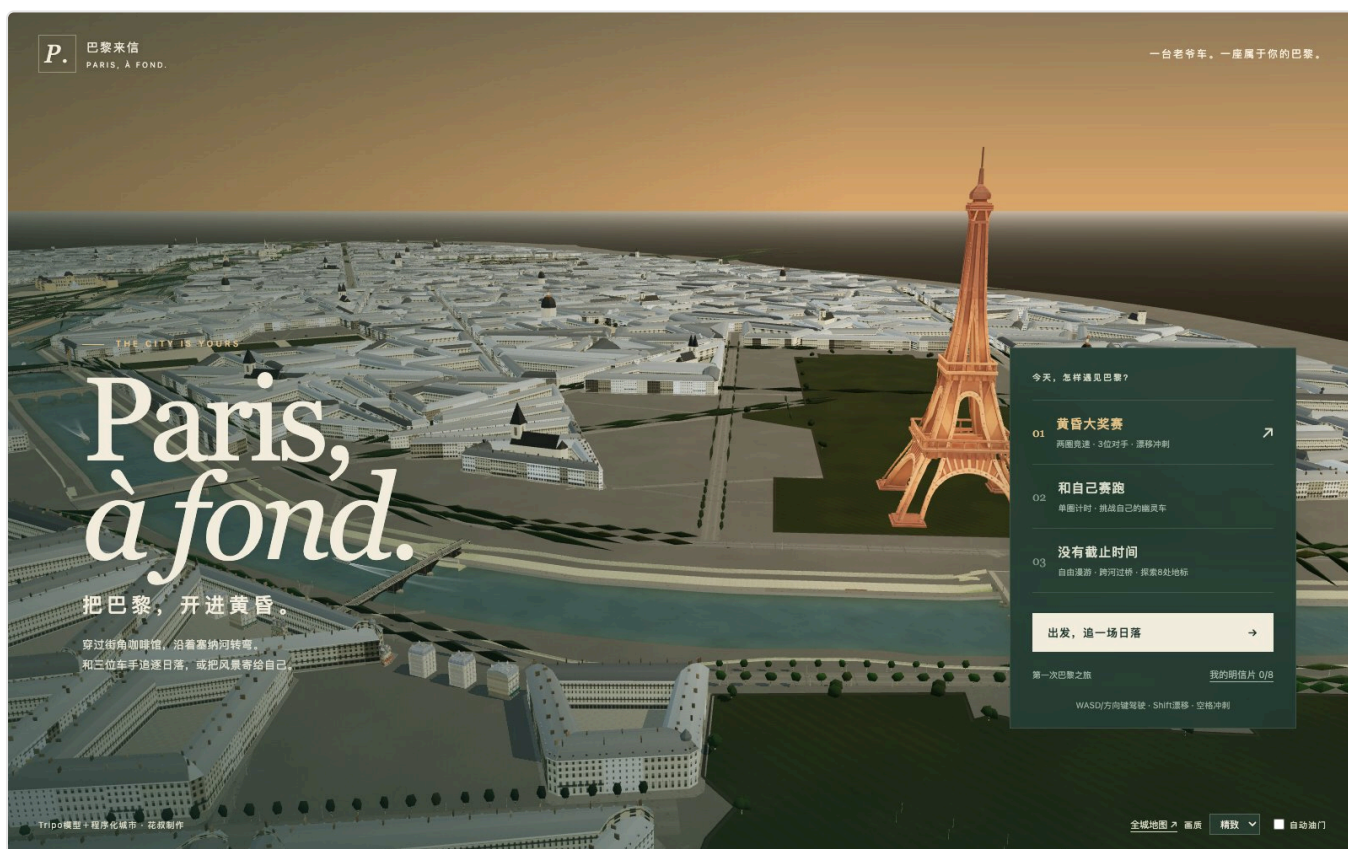
这条对任何类似的沙盒环境都成立，不只对那一家。

第二类是子路径托管。B站Toy这类平台把页面跑在 `/toy/<slug>/` 下面，绝对路径的资源引用会404、根绝对跳转会跳错。包传得上去、审核也可能过，但打开是坏的。我的发布包因此全部用相对路径。

Toy的发布动作本身是两段式的：先不加确认参数跑一次，拿到预览链接自己在浏览器里检查，确认没问题再提交审核。预览就是那道闸门，别一步到位。



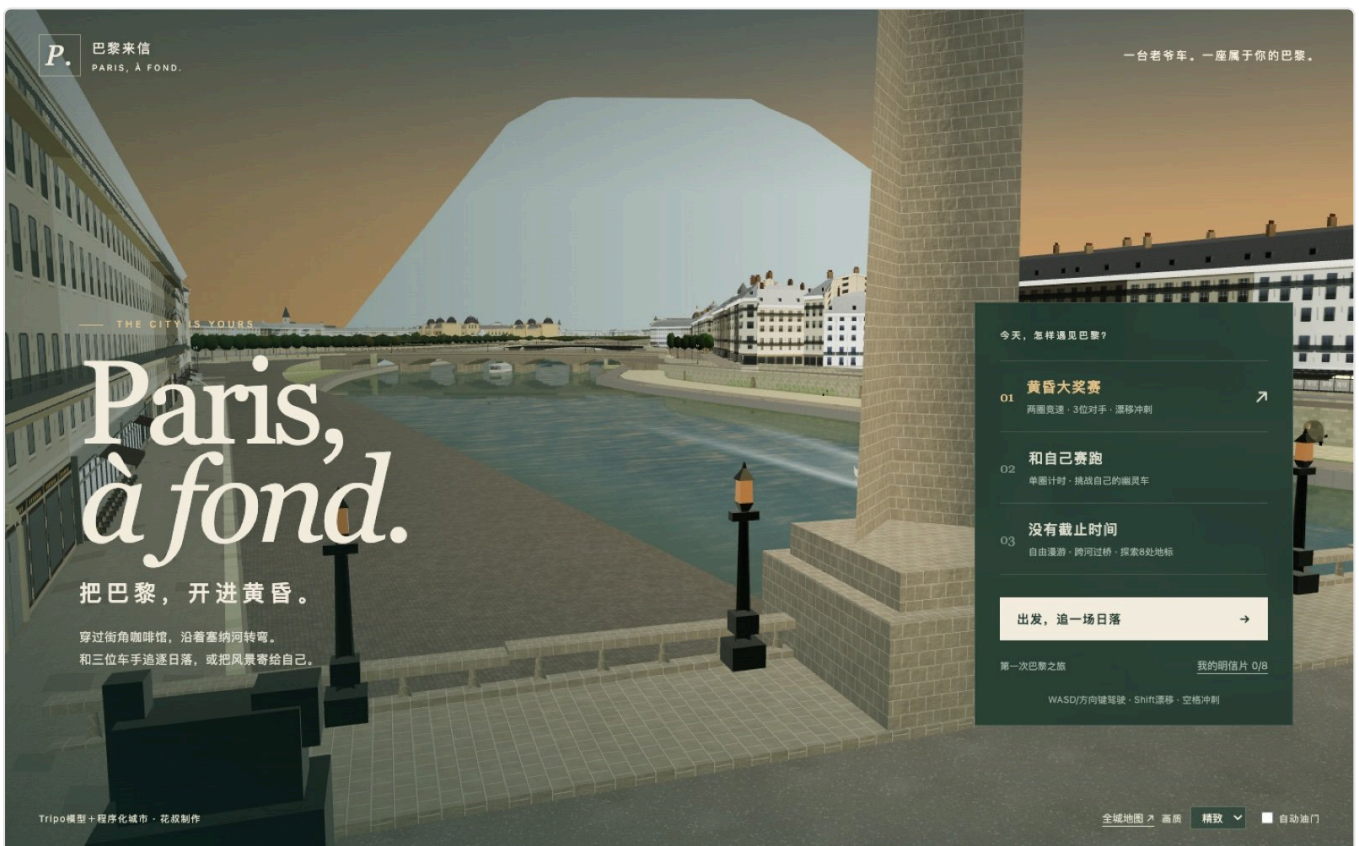
巴黎demo按Tab打开的全城地图。1375个程序化街廓加24座桥加约6975米河道，Tripo资产只有沿街那一圈建筑、车辆和地标。地图包围盒约6050×5340米，不是可通行面积。



铁塔是Tripo生成的（1.9万三角面，天生就低，不需要重拓扑），底下的街道和河堤是程序化城市层。这张是2026-09-14复核时重拍的，下一张来自发布前的QA自检集。



桥面高度是按实际桥面网格取样缓存的，不是套街面 `y=0`。这条是接旧城市层时必须做的适配之一，不做的话车会从桥上掉下去。



水面反射由城市层生成。低画质档会把反射直接关掉（`refl.dead = low`），这是我唯一按画质档做取舍的地方。

混搭：Tripo和程序化不是二选一

最后一条，也是我认为对做游戏最有用的一条。

巴黎城里近景行道树用的是Tripo生成的GLB（两款巴黎七叶树，绿叶和秋黄交替，近景实例配额110；早期是36株、用的还是另一棵梧桐，换树种和提配额的原因都在 § 18），中景、远景和公园的两万株保留程序化球冠。原因很实在：近景要细节，远景只需要「那儿有个绿色的东西」，而AI模型的面数成本撑不住两万株。

这笔账我在巴黎城里同机位同画质量过，开着沿街14栋Tripo建筑对全程序化，代价是多17个draw call、多15%三角面，帧率一帧没掉。完整的两组读数在 § 13。

接Tripo树的时候还有个细节值得记：城市层给的 `c.h` 是「冠高」，而GLB树的宽高比接近1比1（整树包围盒），程序化树是宽8比高11的瘦高个。按高度等比放大到2倍时冠幅会变成14米，而行道树间距才8米，整排会糊成一片。最后 `heightScale` 取1.2，冠幅落在8.4米。



星光游乐园的Three.js版本，后来这条线的最终形态转到了Unity（见 § 14）。两版共用的是一部分巴黎道具和木马，设施是两套：这一版的摩天轮、飞椅、茶杯都是代码搭的，Unity那一版才换成Tripo生成。

接Tripo资产的检查表

检查项	不做会怎样
设了 <code>scene.environment</code>	白底上金属件渲成死黑
环境贴图有暗部	画面发白、没有重量感
白底场景没开ACESFilmic	白底被压成灰

检查项	不做会怎样
按包围盒归一化并把底面压到地面	每件资产都只有1米高
SkinnedMesh不用 <code>Box3</code> 量尺寸	模型被缩到只剩一双鞋
用 <code>SkeletonUtils.clone()</code>	所有实例同手同脚
每个实例一个 <code>AnimationMixer</code> 并错开相位	整排人像仪仗队
切掉动画的位移轨	循环回起点时人会弹回去
贴图降到 1024^2 (特写 2048^2)	显存5 GiB, 卡顿且查不出原因
P2.0的件只当它有base color	一路去找一个不存在的法线图
接P2.0前先算显存 ($8192^2 \times 26 = 9.3$ GB)	纸上就能算出来的崩溃, 非要跑一次才知道
朝向按资产类别分别定, 靠六视图读	车横着开、楼拿素墙对着马路
旋转烘进顶点, 不写节点矩阵	模型静默错位, 控制台干干净净
自写解析器认 <code>normalized</code> 标志	包围盒读成32767倍, 缩放算出32783
量帧时间用GPU timer query, 而且串行跑	量到的是vsync, 或者是隔壁进程
自检截图前确认加载遮罩已淡出	对着一张蒙了白纸的图调材质
页面走http不走 <code>file://</code>	GLB被CORS挡住
资源全用相对路径	子路径托管上白屏或404

§13 Blender：什么时候该进，什么时候别进

Blender — When It Earns Its 300 MB

同一个资产跑三条路径的实测对比、一次失败得很彻底的自动权重绑定，以及几个不打开GLB的JSON就永远发现不了的坑。结论可能和你想的不一样。

先说结论

「Tripo出的模型要不要进Blender」这个问题，我一度以为答案是「当然要，专业流程都这么干」。跑完对照实验之后，我的答案变成了三句话。

只做减面和压缩的话，Blender没有优势。同一个资产，Blender产出6772面、365 KiB，命令行产出6781面、369 KiB，肉眼和数字上都等价，而命令行不需要那三百兆的依赖。

Blender在这条管线上有两件事最值得它那三百兆，头一件是让死的资产活过来。Tripo直出的静态模型是0个skin、0个关节，任何关节动画都做不了；Blender一秒就能给它加上骨架。第二件是格式转换：P2.0带贴图的产物只给FBX，浏览器只吃GLB，而我这台机器上当时装了的、能把FBX读进来再吐成GLB的，只有它，这条在本节最后展开。

但在Tripo的资产上，Blender的自动权重是不成立的。转一下脊柱整个人就倒了。这条是我实测出来的，下面有图。

所以2026年9月这个时点，我的真实推荐是：角色要动，走Tripo自带的绑骨（见 § 06）；Blender留给格式转换、拆件、精修、烘焙和hero资产。

E0：同一个资产，三条路径

实验设置尽量干净：同一个案例 `building-03-boulangerie.glb`（奥斯曼面包店，16696面/ 568 KiB / 512²贴图），环境是Blender 4.4.3无头执行加 `@gltf-transform/cli 4.5.0`，耗时用 `date +%s` 记墙钟。

这件输入不是Tripo直出的原件，是A组那件在本地跑过一道 `gltf-transform optimize` 之后的512²版，也就是附录A的B组；Tripo直出的原件是16874面、1.49 MiB、1024²贴图。三条路喂的是同一个输入，所以对照本身不受影响，但引这几个数的时候别把它当成「Tripo刚出炉的样子」。（本节的文件体积一律按1024进位写成KiB / MiB，和附录A那张逐件表用的是同一把尺子。）

模式B，Tripo直连引擎，两条命令：

```
gltf-transform simplify in.glb tmp.glb --ratio 0.1 --error 1
gltf-transform optimize tmp.glb out.glb --compress quantize --texture-size 512
```

减面3秒、压缩3秒，合计6秒。产出6781面/ 369 KiB。

模式A，进Blender，一段无头脚本：


```
bpy.ops.import_scene.gltf(filepath=SRC)
m = obj.modifiers.new('dec','DECIMATE')
m.decimate_type = 'COLLAPSE'
m.ratio = 6781/16696
bpy.ops.export_scene.gltf(filepath=DST, export_format='GLB', export_apply=True)
```

导入0.2秒、减面加导出0.1秒，含Blender启动的墙钟2秒，后续压缩同模式B再3秒。产出6772面/ 365 KiB。

	面数	体积	耗时	依赖
模式B (gltf-transform)	6781	369 KiB	6s	一条npx命令
模式A (Blender)	6772	365 KiB	2s + 3s压缩	Blender约300 MB

差异在千分之一量级。「必须进Blender」这个迷思，在减面这件事上是假的。

模式C是混搭，不是二选一。我在巴黎城里同机位同画质跑了两个变体：开着沿街14栋Tripo建筑，和把它们全关掉。

变体	draw calls	三角面	帧时间	加载
C1混搭（14栋Tripo建筑+程序化城市层）	233	2,810,450	16.7 ms	2039 ms
C2全程序化	216	2,447,388	16.7 ms	1971 ms
差额	+17	+36.3万 (+15%)	0	+68 ms

用15%的三角预算，买下赛道沿线全部的高细节建筑，帧率一帧没掉（两个变体都锁在vsync 60 fps满帧）。这就是混搭的量化价值。

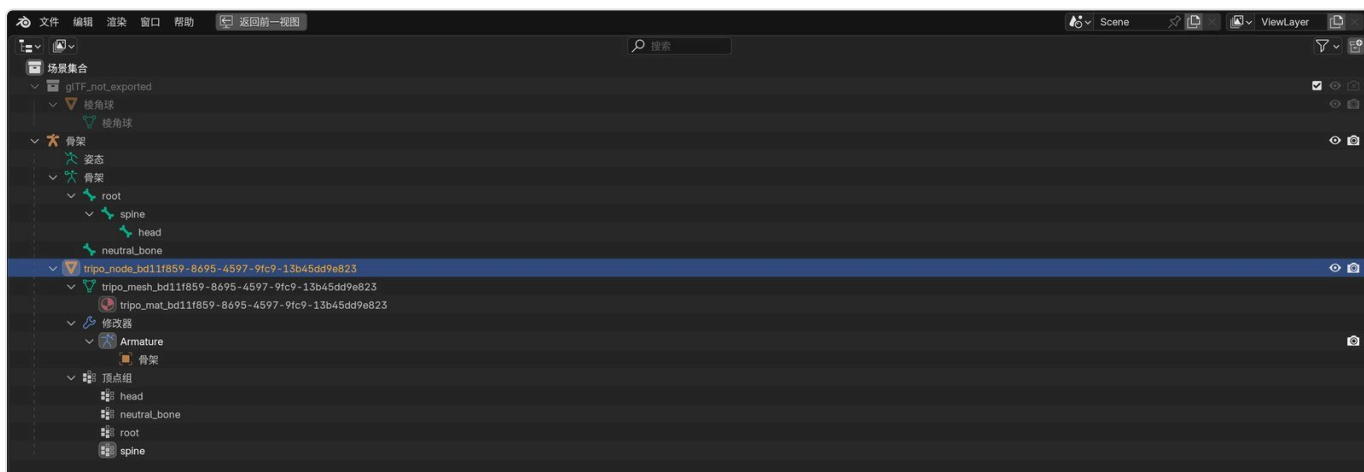
Blender不可替代的那一件事：绑定

用 `pedestrian-01-man-suit.glb`（静态行人，和E0一样取的是B组那件512²版，不是Tripo直出的原件；不过「0个skin、0个关节」这一条A组原件同样成立，降一道贴图不会凭空长出骨架）补测，Blender侧脚本建一条三骨骼链 `root → spine → head`，用 `ARMATURE_AUTO` 自动权重，一秒跑完。

	skins	joints	animations	nodes	体积
绑定前	0	0	0	1	443 KiB
Blender绑定后	1	4	0	6	815 KiB



Blender 4.4.3里打开 `modeA-rigged.glb`。这是那件静态行人经Blender自动权重绑定之后的样子，模型中间那条竖着的橙色线就是全部骨架。注意左上角的对象名 `tripo_node_bd11f859-...`，这是Tripo导出GLB的命名规则。



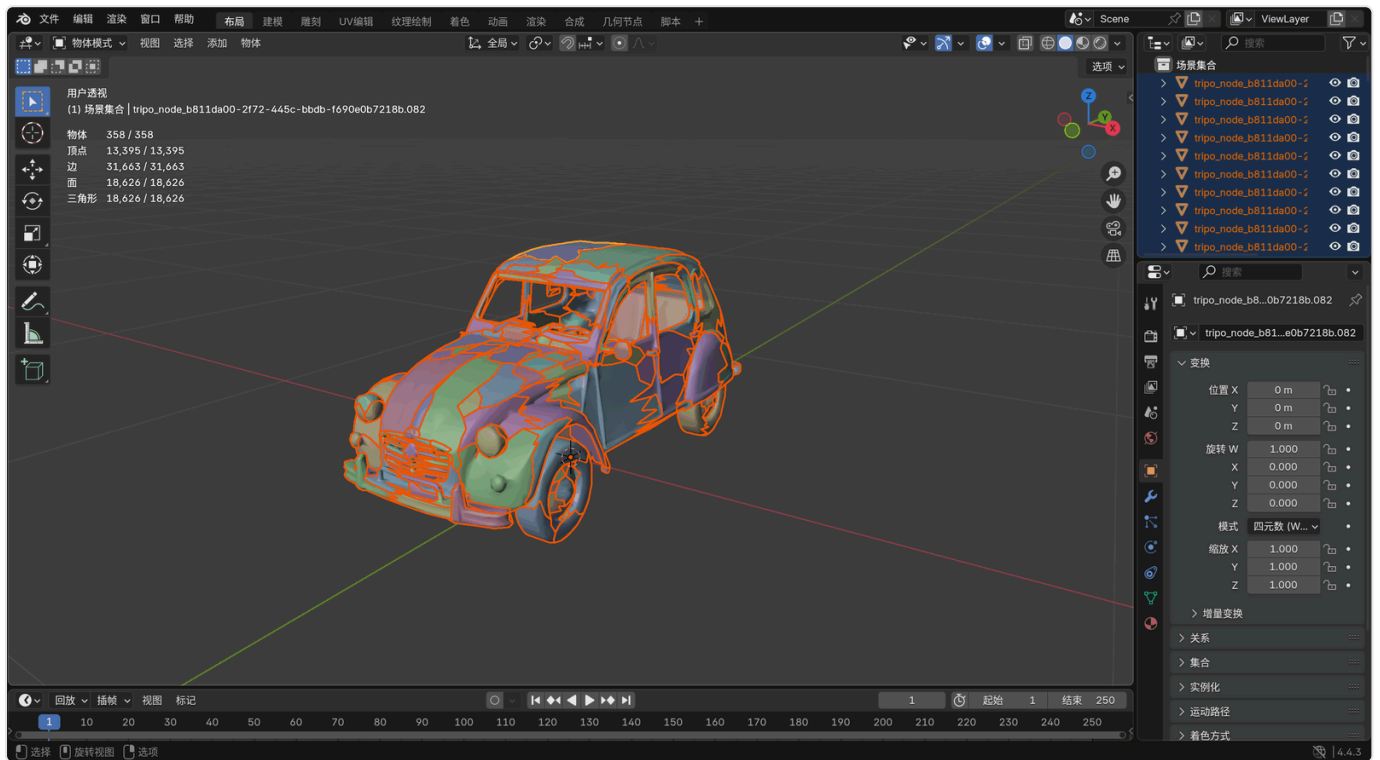
同一个文件的大纲视图（已裁去下方空白）。骨架只有四个：`root` → `spine` → `head`，外加一根 `neutral_bone`；顶点组也是这四个。上面那个 `glTF_not_exported` 集合里的「棱角球」是glTF导入器留下的辅助物体，遍历网格时要排除掉它。

写脚本遍历Tripo导入的场景时，记得跳过 `glTF_not_exported` 集合。不跳的话它会被当成一个真实网格参与计算，面数、包围盒全都会算错。我的脚本里是这么判的：`if any(c.name == 'glTF_not_exported' for c in o.users_collection): continue`。

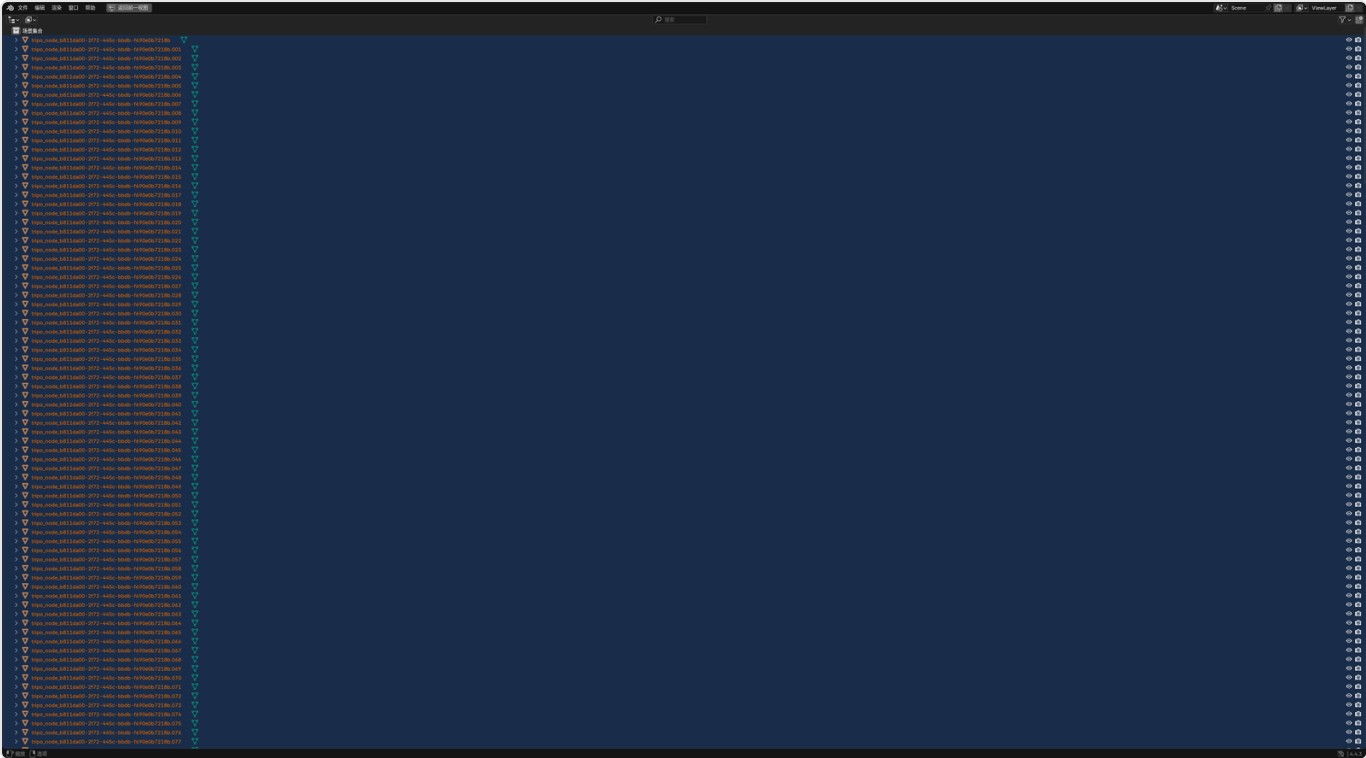
E3：连通分量拆不出轮子

在做「车轮能不能转」这件事之前，我先试了最直觉的路：用Blender按连通分量把网格切开（`mesh.separate(type='LOOSE')`），看能不能得到一个可以独立旋转的轮子。

资产	原始面数	切出块数	最大块
citroen-2cv	18076	358	431面
car-01-taxi	18083	262	655面
car-02-cargo-tricycle	18490	569	320面



同一辆2CV按松散块拆开之后，Blender给每块碎片一个随机物体色、加一圈橙色描边。视图统计角标写着「物体358 / 358」，整车就是这么碎的。拆分本身只花了0.2秒。



同一状态的大纲视图，一屏78条，全是同名 `tripo_node_...` 加序号。没有一个叫「轮子」或者「车门」，这就是「几何连通分量不等于语义部件」最直观的样子。界面缩放临时调到0.5才放得下，没有写进偏好设置。

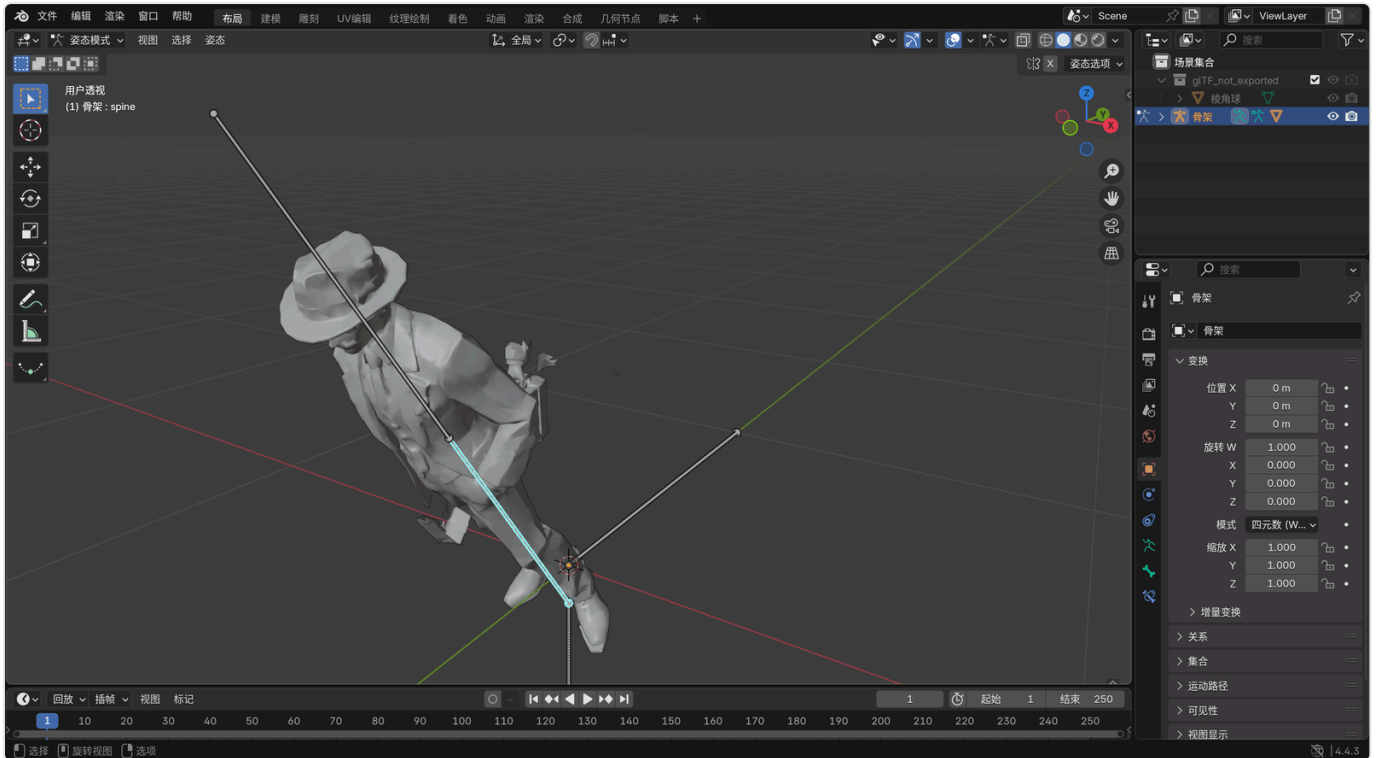
拆不出来。Tripo的输出不是一整块干净网格，而是几百个100到650面的碎片。按连通分量切只能得到这些碎片，得不到「轮子/车身/车门」这种语义部件，因为轮子和车身在几何上是糊在一起的。

这条结论解释了两件事。我那个游戏里车轮转不了，根因是源模型里根本没有可分离的轮子，跟我做没做没关系。而Tripo自己那个「智能拆分」功能之所以是刚需，是因为它做的是语义拆分（按物体含义分件），这是几何算法做不到的事。拆件的实操和实测结果在 § 05，这里不重复。

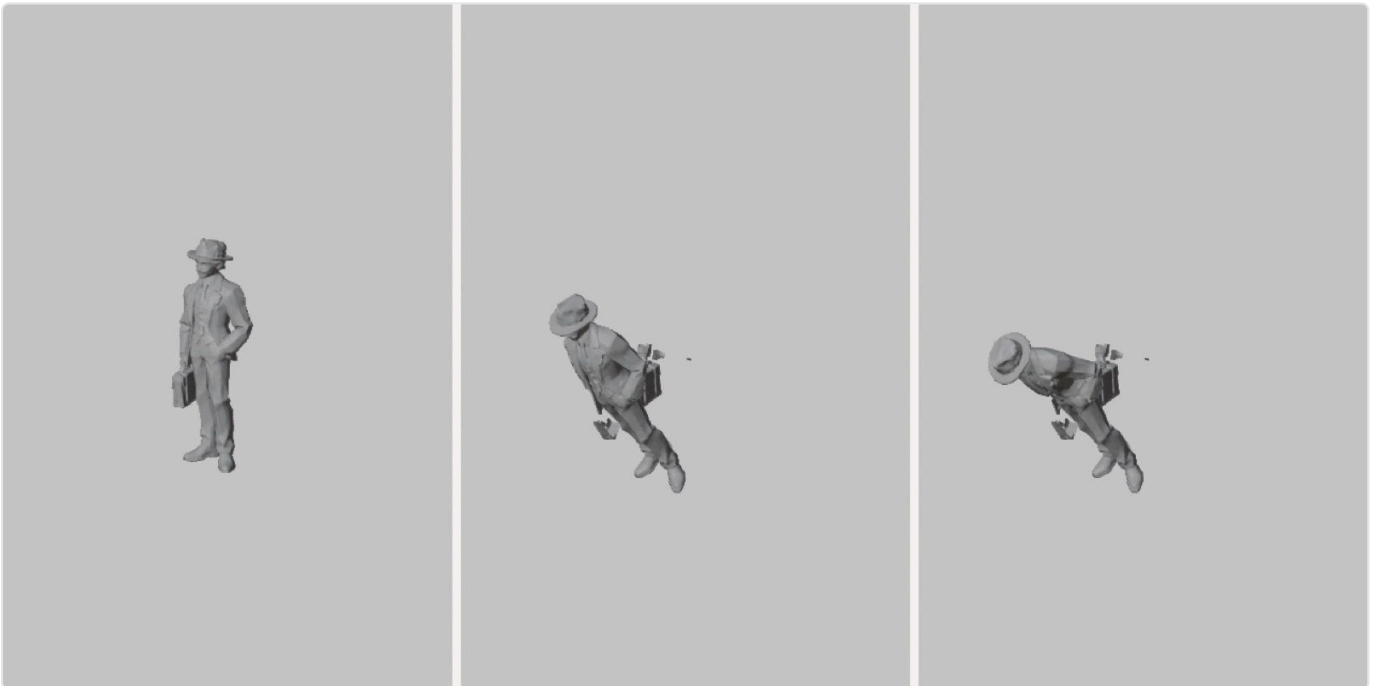
E4：自动权重绑上了，但权重是错的

绑定生效不等于绑定可用。我把E0里绑好的那个文件重新导入Blender，进姿态模式依次转 `spine` 和 `head`，渲三张图。

姿势	结果
A静止	正常站姿（戴帽、西装、手提公文包）
B把 <code>spine</code> 转35度	整个人向前倾倒，不是上半身相对下半身弯曲，腿也被spine的权重带走了
C再转 <code>head</code> 30度	整个人翻倒，另有一小块碎片飞离主体



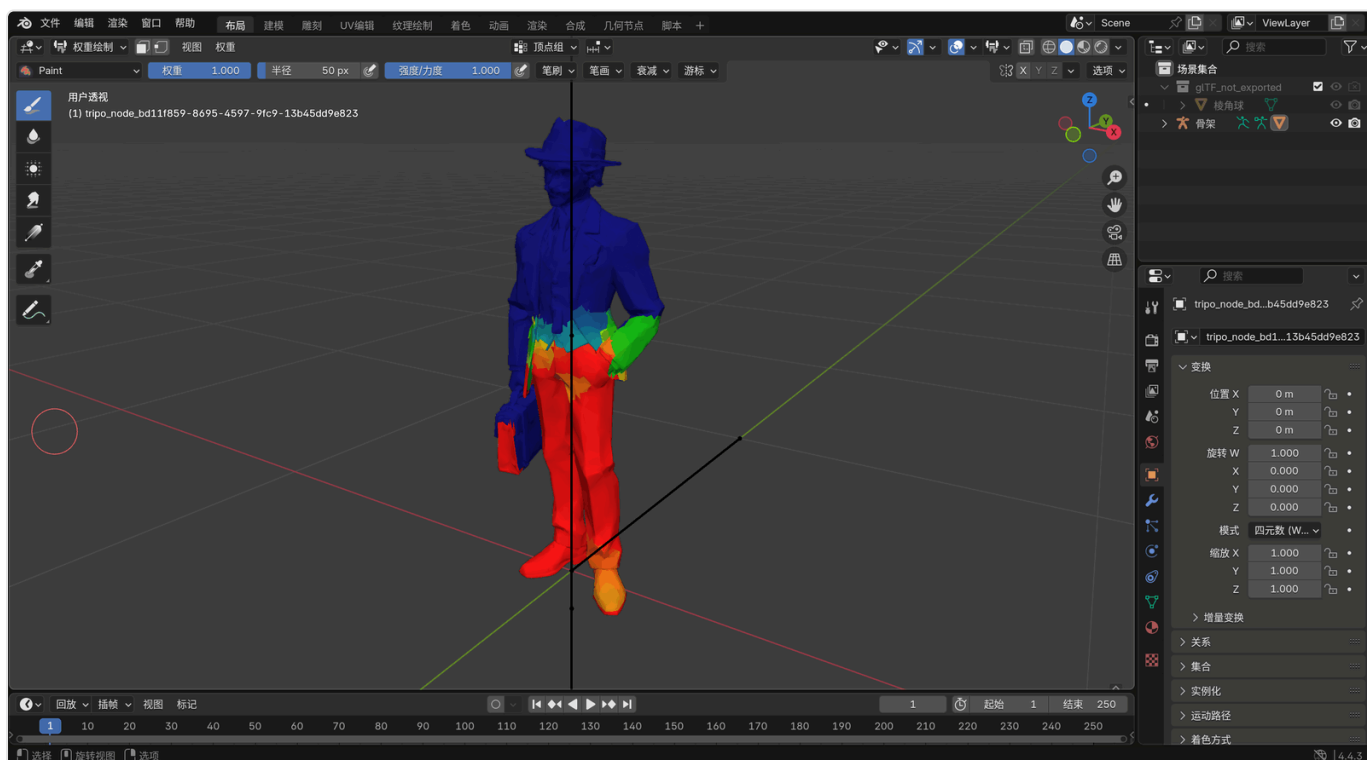
姿态模式下把 spine 转35度的现场。左上角写着「骨架: spine」, 而画面里整个人已经不成人形。这不是渲染问题, 网格确实跟着骨骼走了, 只是权重把不该带的部分一起带走了。



E4实验的三张对比, 左起: A静止、B转 spine 35度、C再转 head 30度。Blender workbench渲染, 脚本 E4-绑定质量/pose_render.py, 2026-09-10。原图顶部那条中文标签渲染时缺字体、是一排方块, 已裁掉。



带材质的版本，左A静止、右B转 `spine` 35度。用纯灰模看不清楚的地方（比如右边那块飞离主体的碎片）在这里更明显。顶部标签同样已裁。



权重绘制模式，当前活动顶点组是 `spine`。红色区域是被这根骨骼完全带动的顶点。看它从胸口一路铺到腿上，就知道为什么转脊柱会把腿一起带走。

根因和E3是同一个。模型是几百个碎片拼成的，没有连续的蒙皮拓扑，`ARMATURE_AUTO` 算出来的权重自然对不上；碎片还会在变形时脱离主体。

独立评测里对AI生成网格的通用描述是「多边形分布均匀、边环不贴合解剖结构」。碎片化正是「没有边环结构」的表现，而边环恰恰是自动权重赖以工作的东西。

换成Tripo自带绑骨，同样的动作

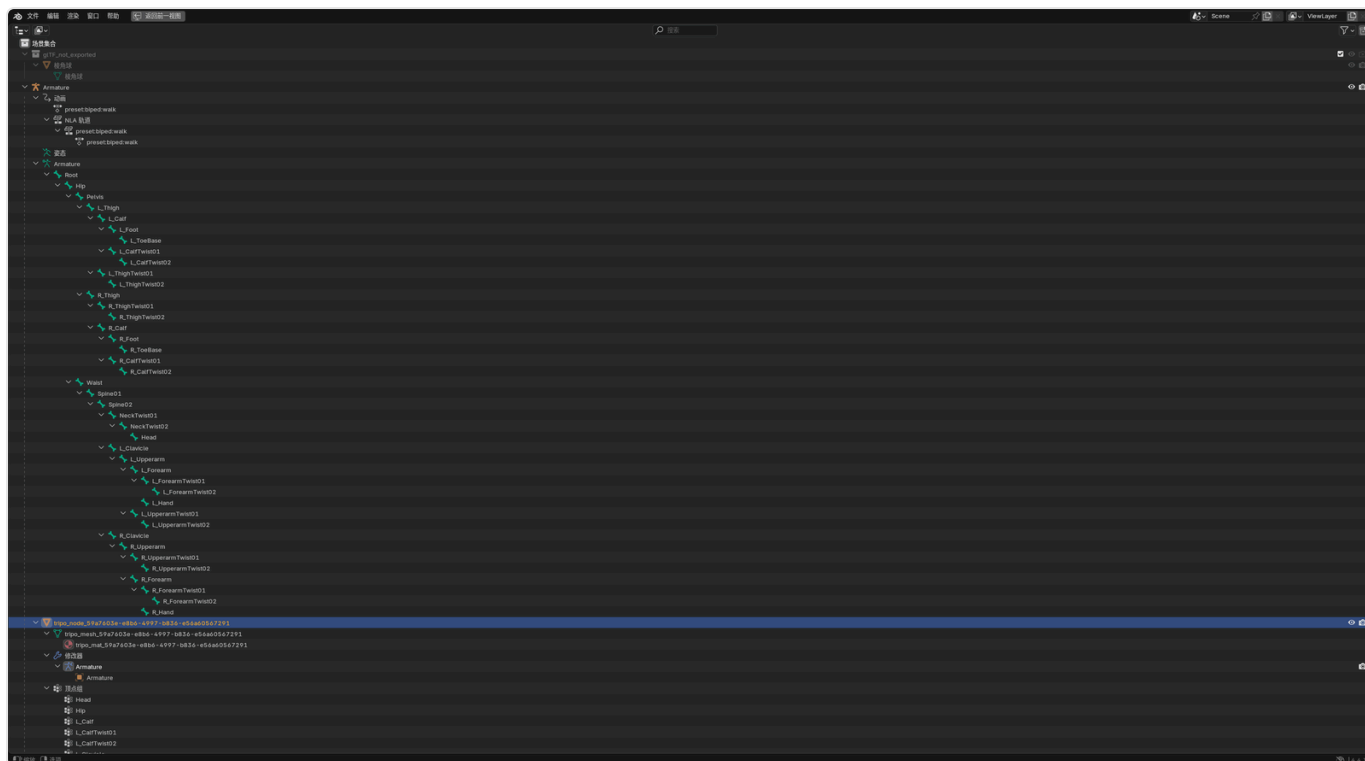
为了看清楚差在哪，我把一个Tripo自动绑骨出来的角色也导进Blender，做完全相同的动作。

注意

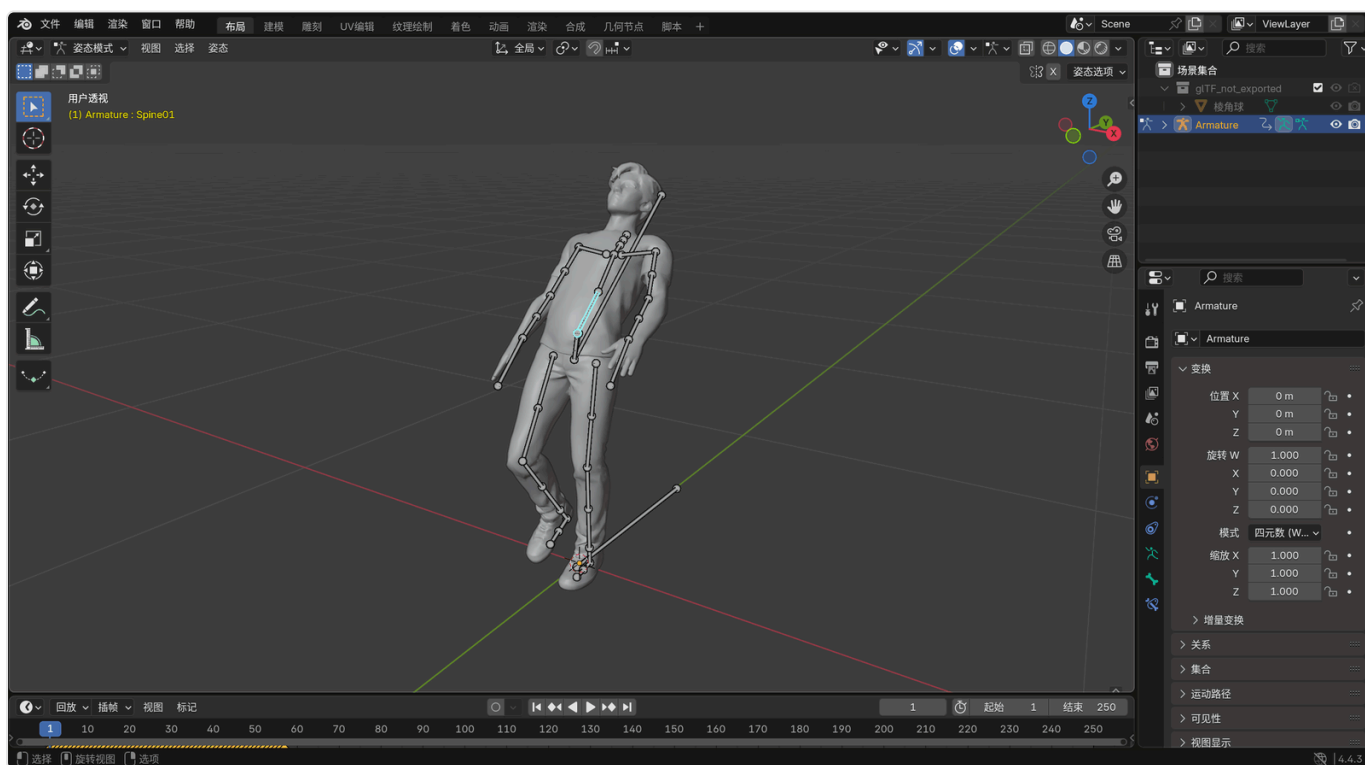
先说清楚一件事：这两个角色不是同一个人。上面那组是巴黎那个戴帽子提公文包的行人（pedestrian-01-man-suit），这一组是游乐园那个穿白T恤的主角（walker-rigged）。我核过两个GLB内部的对象名，uuid不一样，确实是两个模型。所以严格讲这不是同一网格的AB对照，它对照的是两种绑定方式在Tripo这一类网格上的结果。要看同一个模型两种绑定的严格对照，去 § 15那个可以拖滑块的实时页面。



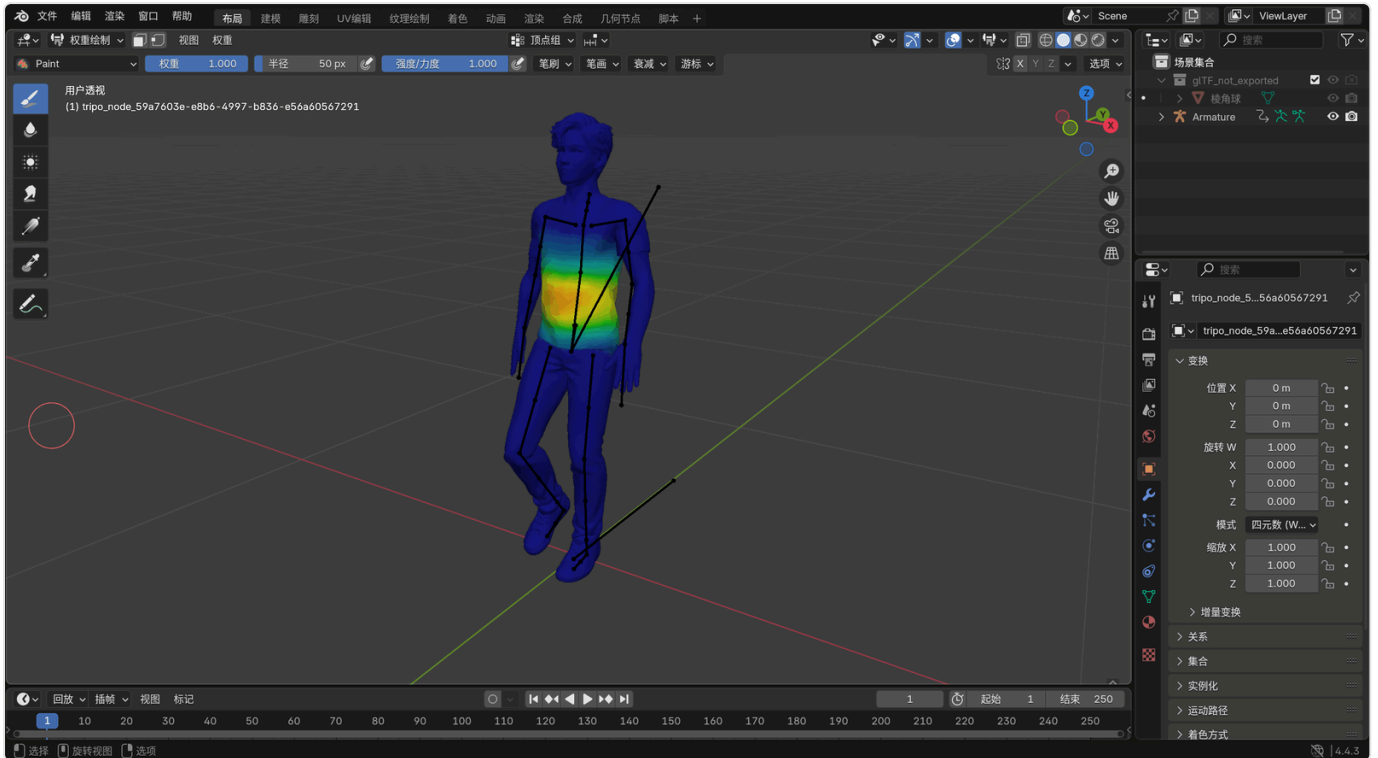
Tripo自动绑骨（20积分）出来的角色，在Blender里打开。骨架是完整的人体骨架，橙色线条从躯干延伸到四肢。这就是游乐园那个主角，它的白T恤加浅灰裤后来在Unity里给我惹了个大麻烦，见 § 14。



同一个文件的大纲视图，这是我最喜欢的一张对照图。Root → Hip → Pelvis → L_Thigh → L_Calf → L_Foot → L_ToeBase，还有 CalfTwist、ThighTwist 这类扭转骨，Waist → Spine01 → Spine02 → NeckTwist01/02 → Head。动画那一栏里挂着 preset:biped:walk，是我在Studio里勾的那段行走。



把 Spine01 转35度。人还是一个完整的人，只是躯干后仰。同样的动作在四关节版本上让整个人塌成了一堆几何体（见前面那张塌掉的图）。



Tripo版本的权重绘制，活动顶点组 Spine01。权重集中在躯干下段并向上衰减，腿部是蓝的。对照前面那张铺到腿上的红，差别一眼可见。

核心建议

4关节和41关节不是数字差距，是「一根棍子」和「一副骨架」的差距。这两组截图是本书里我最推荐你自己重做一遍的实验：两个GLB都在资源包里，导进Blender、进姿态模式、转一下脊柱，两分钟就能看到。

那什么时候该进Blender

把该做的和不该做的分开写，比一句「看情况」有用。

推荐

该进Blender

- **hero资产精修**：主角、近景特写、会被镜头怼脸的东西。AI网格在手指、耳朵这类复杂区域有n-gon和捏缩顶点，UV有接缝，这些只能人工修
- **拼装与合并**：把分开生成的头发、头、身体合成一个角色，这是X上做角色的人普遍在走的路
- **烘焙**：把高模的细节烘到低模的法线贴图，用低面数换回细节
- **物理与动力学**：摇摆物理、布料、头发，Tripo当前给不了
- **面部表情**：blendshape这一层

不推荐

别为这些开Blender

- **只做减面**：E0已证明产出等价，命令行更快
- **只做贴图降采样**：glTF-transform resize 一条命令，见 § 12
- **给已经是GLB的件降8K贴图**：直接改glTF的JSON和BIN更安全，Blender一个来回反而会动到不该动的量，见 § 12
- **批量清洗**：25个资产131秒跑完，平均5.2秒一个，全自动
- **指望自动权重救场**：E4已证明在Tripo资产上不成立
- **拆语义部件**：E3已证明连通分量拆不出来，要用Tripo的拆件（见 § 05）

先过一道：你拿到的这个GLB，Blender可能根本打不开

上面所有实验用的都是带贴图的那批文件，一导就进。但我后来撞上了两种导不进去的情况，问题都出在文件本身，不在Blender。

第一种：文件名里带 meshopt 的那个变体。

```
Extension EXT_meshopt_compression is not available on this addon version
```

Blender 4.4的glTF导入器不支持这个压缩扩展，报完这一行就结束，模型一个顶点都进不来。我在两条完全不同的线上各撞了一次：一次是做两种模式对照实验时去拿几何数据，一次是渲Blender演示素材时。**两条线撞的是同一件事，说明它不是偶发。**

怎么避开，两条路：

1 从详情页的「导出」弹层换成FBX

下来是个zip，里面是 `tripo_convert_<id>.fbx`，未压缩，几何和那个GLB逐数吻合。导出这个动作不扣积分，我盯着余额看过，前后一分没变。

2 本地先把压缩解掉

`npx --yes @gltf-transform/cli copy in.glb out.glb`，走一遍完整读写流程就把meshopt脱掉了。这一招和 § 14里让Unity能导入的那一招是同一条。

第二种：为了省积分关掉贴图生成的那种。这条是个陷阱，值得专门说。

v3.1这条路把「贴图生成」关掉，价格会明显往下走。听起来像是「我先出个白模，贴图以后再说」，但实测的产物是这样的：

```
GLB 顶点属性：只有 POSITION 和 NORMAL
materials = 0   textures = 0   images = 0
```

连UV都没有。UV是把二维贴图贴到三维表面上的那套坐标，没有它，任何贴图都无从贴起。所以「以后再补贴图」这条路是不通的，要贴图只能回Tripo重跑一次。

核心建议

怎么用这个模式才不亏：确定这一件只当碰撞体、只当远景白模、只用来量尺寸，那就放心关；只要还有一丝「将来可能要看它的表面」，就带着贴图出。判据是「这件东西会不会被镜头看到表面」，不是「现在需不需要」。

Tripo的GLB导进Blender之后长什么样

在写脚本之前，先知道你会拿到什么。这几条看起来琐碎，但每一条我都是因为没预料到才多花了时间。

命名带uuid。一个模型进来是三件套：`tripo_node_<uuid>`、`tripo_mesh_<uuid>`、`tripo_mat_<uuid>`，uuid是那次生成任务的标识。好处是不会重名，坏处是脚本里没法按人类可读的名字找对象，得按类型或顺序找。前面那两张大纲视图截图里能看到完整的名字。

多一个 `glTF_not_exported` 集合。里面通常躺着一个「棱角球」，是glTF导入器的辅助物体，不是你的模型。它不会被再次导出，但会被 `bpy.context.scene.objects` 遍历到。

材质是PBR而且带金属度。Blender里看着正常，因为Blender的默认HDRI环境把它照亮了；一进Three.js或者白底场景就变黑（见 § 12）。Blender里看着对，不代表引擎里看着对。

整体高度接近1.0单位。这条和引擎侧是同一件事，见 § 14坑一。在Blender里表现为：导进来那个模型在默认网格上小得像颗豆子，你会下意识去缩放视图，然后忘了它本身的尺寸也是要处理的。

Blender、命令行、混搭的总对照

	模式A（Blender）	模式B（直连）	模式C（混搭）
减面产出	6772面/ 365 KiB	6781面/ 369 KiB	—
减面耗时	2s + 3s	6s	—
绑定能力	有（skin / joints）	无	无
空间成本	—	—	+15%三角/ +17 calls
依赖	Blender 4.4.3（约300 MB）	gltf-transform	两者皆无
什么时候非它不可	角色、需要动画的任何东西；本机已有Blender时，FBX转GLB走它最省事	场景资产的量产清洗	大规模重复元素

压成三句：要动的东西绕不开A；只要摆着好看的东西B更快；要摆满一整个城市，C是唯一答案。

那还能不能在Blender里绑好Tripo的资产

E4证明的是「直接对Tripo直出的静态网格跑自动权重」这条不成立，它没有证明「Blender绑不了Tripo的东西」。这两句话差别很大，我不想把前者说成后者。

按E3和E4的根因往回推，有条路在逻辑上是通的：碎片化是自动权重失败的原因，那就先把拓扑变连续，再绑。可以先在Tripo里做一次重拓扑（10积分，产出四边面，见 § 04），或者先做语义分件把部件分开（见 § 05），拿到更规整的网格之后再进Blender。

注意

上面这条是推断，我没有实测。写在这里是因为它是下一步该做的实验，不是因为它已经成立。真要做的话，验证方式和E4完全一样：导进Blender、进姿态模式、转 spine 35度，看人还是不是一个人。两分钟就有结论，不需要相信任何人的说法。

无头Blender：不开界面也能干活

上面那些实验没有一个是手点出来的，全部是 `--background` 模式跑脚本。对不会写代码的人来说，这反而是更省事的路：你不需要学Blender的界面，只需要让编程Agent写一段脚本。

```
blender -b --python modeA.py -- in.glb out.glb
```

脚本里三件事：`bpy.ops.import_scene.gltf` 导入、加修改器或建骨架、`bpy.ops.export_scene.gltf` 导出。要出图的话再加一步渲染。这一节所有Blender截图也是脚本截的，用的是Blender自带的 `bpy.ops.screen.screenshot`，整窗口一次拿下。

让Agent写这种脚本，我的经验是把三件事在需求里说死，剩下的它能自己搞定。

1 Blender的可执行文件在哪

macOS上是 `/Applications/Blender.app/Contents/MacOS/Blender`，不是 `blender`。不说清楚它会写一个在你机器上跑不起来的命令，然后你们俩一起调半天PATH。

2 输入输出用绝对路径，参数从 `--` 后面取

Blender会吃掉 `--` 之前的所有参数，脚本里要 `sys.argv[sys.argv.index('--')+1:]` 才拿得到你传的那些。这是个纯约定，Agent不一定记得。

3 让它把结果打印出来，而不是只写文件

我的脚本每一步都 `print` 一行（`PREP_LOOSE parts=358`、`SHOT OK ...size=757032`），因为无头模式下你唯一能看见的就是日志。只写文件不打印，出了事你连它跑到哪一步都不知道。

还有一条边界要先知道：`--background` 下没有OpenGL上下文。`bpy.ops.render.opengl()` 会直接报 `Cannot use OpenGL render in background mode`，所以视口叠加层那一整套都拿不到：线框overlay、左上角的统计信息、Loop Cut的黄线，一个都没有。要「能认出这是Blender界面」的画面，只能开前台录屏；无头能给你的是干净的渲染结果，没有系统边框、没有鼠标、没有菜单栏。我这一节的线框图是用Wireframe修改器把每条边变成真几何渲出来的，面数标签是事后用PIL画上去的。先知道这条边界，就不会花半天去调一个不存在的开关。

还有一条是验收习惯：截图脚本报的OK，判据要是文件大小而不是退出码。我的 `shot()` 函数里写的是 `os.path.exists(p) and os.path.getsize(p) > 10000`。Blender在无头或者窗口没画完的情况下能写出一个几百字节的空PNG，退出码照样是0。

注意

Blender约300 MB的依赖不是零成本。如果你的管线要跑在CI或者别人的机器上，多一个Blender就多一份「装没装、版本对不对」的麻烦。只有当你真的需要上面「该进」那一列里的能力时，这三百兆才值得。

FBX转GLB：本机已有Blender，这道转换就走它

P2.0带贴图的产物只给FBX，而浏览器只吃GLB，中间必须过一道转换。我选Blender的理由是「我这台机器上当时装了的只有它」，不是「只有它做得到」。后面那句我没有资格说，因为要说它得先把别的方案一个个试过，而我一个都没试。

我当时探的那一圈是这样：`assimp` 没装、`pygltflib` 没装、`gltf-transform` 虽然好用但它只吃glTF，读不了FBX。而Blender 4.4.3就在 `/Applications` 里，`--background --python` 可跑，不用再装一个新东西、不用再验一遍它认不认这批文件。

注意

专门做这件事的工具是有的，我只是没测。FBX2glTF那个独立二进制、Assimp、Autodesk自家的FBX SDK都能做FBX转glTF，Unity和Unreal更是导进去再导出就行。它们快不快、轴向和贴图对不对得上，我一个都没跑过，所以不替它们说好话，也不替它们说坏话。你要是本机没有Blender、或者这条管线要上CI，先去看它们，别为了这一道转换背三百兆。

还有一条更硬的理由：这批FBX的Creator字段写的就是Blender的FBX IO。也就是说文件是Blender导出的，那就用同一个实现读回来，轴向和单位的往返最不容易出岔子。

核心建议

Blender做转换，纯Python只做「真值」和「核验」，两侧互相不信任。纯Python方案自己解FBX再拼GLB其实也做得到，但要自己处理四边形三角化、法线层的 ByPolygonVertex 展开、UV接缝拆点，都是能写对但没必要自己扛的活。分工清楚之后，这条管线就有了一个很好的性质：它能自己证明自己没弄坏东西。

三通道包围盒交叉核验

转换有没有把模型弄坏，最怕的是「看着挺对」。我用了三条互相独立的路径去算同一个世界包围盒：

1 纯Python解FBX

自己读顶点，再手算那个 `Rx(-90)`。FBX里几何顶点存的是Z-up局部坐标，靠Model节点上的 `Lcl Rotation = (-90, 0, 0)` 转进Y-up的世界。

2 Blender侧读

导入后直接读世界包围盒，再按Z-up到Y-up映射一次。

3 解产出的GLB

读accessor加节点矩阵自己算一遍。

三者最大偏差1.0e-08。这个数字的意思是：三条路算出来的是同一个东西，转换没有偷偷改尺度，也没有偷偷改轴向。

为什么盯着包围盒不放，因为真正会把一整座城搞散的就三件事，而它们全落在这一个可测量的量上：Y必须是上轴（引擎里缩放系数是 `目标高度 / size.y`，上轴错了尺寸直接乱套）、长宽高比不能变（等比缩放，比例变了模型就变胖变瘦）、绕Y的朝向要和被替换的旧件一致。而绕Y转90度会让X和Z互换，所以包围盒这一条顺带把朝向也兜住了一半。

	源FBX	产出GLB
文件体积	7,473,036 B（7.13 MiB）	747,724 B（730 KiB）
多边形	5,615（四边形4,185，占74.5%）	—
三角面	9,800（三角化后）	9,800，逐位相等
顶点	5,381	11,617（UV和法线接缝拆点）
贴图	一张8192 ² JPEG，6.81 MiB	一张1024 ² JPEG，308 KiB
包围盒	0.516113 × 0.543457 × 0.991699	偏差1.0e-08

顶点数从5,381涨到11,617是UV和法线接缝拆点的正常结果，不是转坏了。这一条我要标一句：涨了2.16倍我没有再去查是不是拆多了，因为26件合计约30万顶点，在预算内，所以没深究。

注意

FBX里有个看着吓人的字段：**UnitScaleFactor = 100**。实测Blender导入后世界尺寸和几何局部尺寸逐位相同，没有发生除以100这件事。但这是实测出来的，不是保证，所以核验表里那条包围盒断言必须一直留着，不能因为跑对了几次就删掉。



转换产物（上排）和被它替换的v3.1旧出租车（下排），同一组机位渲染的四视图：正面、侧面、顶视、斜45度。--yaw -90 之后四个机位逐一重合，这是「朝向对了」唯一靠得住的证据：包围盒对 +90 和 -90 完全一样，只有渲出来才分得清车头车尾。2026-09-11。

命令长这样，一件一条，跑完脚本自己把核验跑完并打分，全绿才返回0：

```
python3 fbx2glb.py taxi-p2-smartmesh-textured.fbx \
  -o out/car-01-taxi.glb --yaw -90 --dump-texture
```

--dump-texture 会把降采样之后的贴图另存一张PNG，用来肉眼确认没糊没坏。这一步别省，数字全对而贴图糊了的情况是存在的，数字看不出来。

三个不打开GLB的JSON就发现不了的坑

这三个都是我在这条管线上踩实的，共同点是：模型渲出来一模一样，你只有去读那个JSON才知道出了事。

一，`use_backface_culling` 的语义是反的。Blender的FBX导入器默认不开背面剔除，导出成glTF就是 `doubleSided: true`，每个三角形正反两面都要跑一遍片元着色，片元开销直接翻倍。我那38件全中招，而且和巴黎demo里现有的件全不一致。修法只有一行，但符号要摆对：

```
mat.use_backface_culling = not double_sided
# use_backface_culling = True → glTF 的 doubleSided = false
```

薄片类资产（招牌、树叶、旗子）是真需要双面的，那些要单独开。判据是「这件东西有没有厚度」，不是默认值。

二，Blender会在FBX旁边建一个 `<文件名>.fbm/` 目录。那是它解包内嵌贴图的地方，你以为只是读了一下文件，其实它往你的素材目录里写了东西。我的脚本因此先把FBX复制进临时目录再导入，跑完就删。凡是「只读」的操作里藏着写动作，都值得记一笔，因为它会在你完全没预期的时候污染原始素材目录。

三，替换贴图不能「所有槽位都换成第一张图」。我第一版就是那么写的，因为P2.0那件只有一张base color，怎么写都对，看不出问题。换成一件带normal和ORM的FBX，它会把法线图也替成颜色图，然后你会得到一个表面凹凸完全乱掉、但不报任何错的模型。现在的写法是按原图名一一对应，**对不上直接报错退出**。

核心建议

这三条串起来是同一个道理：**GLB是个容器，渲染结果对不代表容器里的东西对**。验收一个转换产物，除了看图，还要去读它的JSON：`doubleSided`、`extensionsRequired`、`primitives[].mode`、每个材质挂了哪几张图。这几项读一遍花不了一分钟，能挡掉的问题都是那种放到后面才爆、爆了还查不出原因的。

一句话

Blender在这套流程里的位置，不是「专业人士的必经之路」，是「AI暂时做不到那一层的补齐」。它补的是绑定、动力学、面部和hero级精修，而不是减面和压缩。**哪一层AI补上了，Blender在那一层的价值就退一步**。Tripo自带绑骨从20积分把41关节交到你手上这件事，本身就是这条边界在移动的证据。

§14 Unity：把资产搭成一座会动的园区

Unity — Ten Traps Between a GLB and a Playable Build

一座游乐园，六座设施，从Tripo的GLB到55秒巡游片、51 MB网页版和191 MB原生可玩版。中间十个坑，其中一个的根因对任何用glTFast的项目都成立。

这条路给你什么，要你付什么

星光游乐园这条线，最早是Three.js版，最后落在Unity上。换过去的理由只有一个：要出片。Unity能给你实时阴影、后处理、定帧渲染和一个能双击就玩的原生应用，这几样在浏览器里要么没有要么很贵。

代价是十个坑。我按踩到的顺序写，前面几个是「资产进不来」，中间几个是「进来了但不对」。编号只到十，但十个之后还有五件同等量级的事——构建成功却启动即崩、角色发白、WebGL体积、出片管线、镜头连错三次——它们各自太大，所以单独起了标题，没挤进那个编号里。

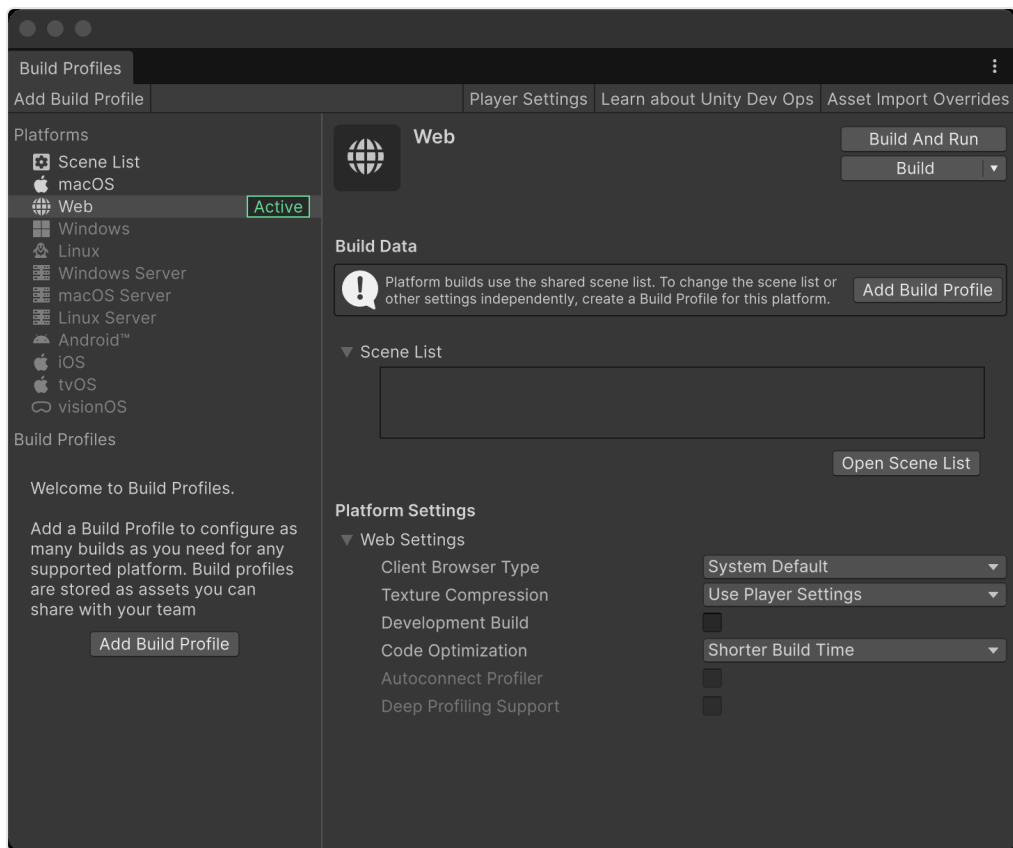
先交代版本，因为Unity 6改了两个名字，照着旧教程会找不到东西。我用的是**Unity 6000.0.83f1**，GLB导入走**glTFast 6.0.1**。

核心建议

还要交代一件同等重要的事：这个工程走的是Unity内置渲染管线（Built-in），没装URP也没装HDRP；后处理用的是Post Processing Stack v2（`com.unity.postprocessing 3.4.0`），代码里那句 `using UnityEngine.Rendering.PostProcessing;` 就是它。本节后面所有跟光和后处理有关的数字——像素光的盏数、Bloom阈值、ACES调色、曝光——都只在这一套里成立。URP和HDRP是另外两套东西：光照上限的算法和配法不一样，后处理也换成了Volume那一套，组件名、参数名、取值范围全都对不上。你要是在URP工程里照着这里的数字调，第一下就会对不上，而且不会有任何报错告诉你原因——所以先看一眼自己工程用的是哪套，再决定这些数字要不要抄。

注意

Unity 6已经没有「Build Settings」窗口了，File/Build Settings... 这个菜单项不存在，取而代之的是File/Build Profiles。而且「WebGL」平台在界面上改叫「Web」了，但代码里 `EditorUserBuildSettings.activeBuildTarget` 返回的仍然是 WebGL。界面一套叫法、API一套叫法，搜教程时两个词都得试。



Build Profiles窗口，Web平台标着Active。这就是Unity 6里取代Build Settings的那个窗口。Unity 6000.0.83f1，2026-09-11实测截图。



星光游乐园入口大道，Unity编辑期离屏渲染，1920×1080，含Bloom和ACES调色。地面的光池不是实时灯照出来的，是画进贴图的，理由见后面第六个坑。

分工原则：这条是整套东西的骨架

在四冲程发动机那轮验证过的判据，在游乐园这里照样成立，而且游乐设施天然就分得开。

用代码（决定机构对错，尺寸必须精确）	用Tripo（决定观感，形体复杂）
摩天轮的圈、辐条、转轴必须是正圆	摩天轮的吊舱
过山车轨道必须闭合、必须平滑	旋转木马整座
旋转木马的旋转轴、飞椅的中柱与伞盖	飞椅、茶杯
茶杯的底盘与自转相位	鬼屋、树、摊位、喷泉
地面、路灯、长椅、树的排布	—

判断标准一句话：这个件错了，别人一眼能看出「机构不对」吗？能，就写代码。这条判据的完整推演在 § 15。

换成给玩家的说法更直白：会被凑到跟前细看的東西用Tripo做，只会远远瞥一眼的，代码画就够了。游乐园这张表就是这条判据的逐行落地：木马、吊舱、飞椅是Tripo，转它们的那个圈、那条轨道、那根转轴是代码。

本轮新生成的Tripo资产六件：旋转木马、摩天轮吊舱、旋转飞椅、旋转茶杯、鬼屋、海盗船，参考图由Codex批量出（8张串行约11分钟，全部成功）。树、长椅、花坛、喷泉、广告柱、摊位沿用巴黎那轮。

核心建议

这一节的构建、渲染、出片是**本机墙钟**，你的机器不一样数就不一样；书里Tripo侧生成、拆件、绑骨的秒数一律是**算子耗时**（算子耗时口径，不含上传与排队；端到端要往上加，两个口径的实测差见 § 01，词条见附录 D）。排管线工期别拿算子耗时去估。



工程里 Assets/Models 的22个GLB，缩略图都已生成。缩略图能出来说明导入成功了，这一点在下面坑二里会变成有一个有用的判据：导入失败的GLB在这里是没有缩略图的。

Tripo自己给模型起的名字可以当质检信号读。它给这批的命名是 ornate gold and red carousel with horses, illuminated details, brass...、red and gold spherical capsule pod with glass windows...、haunted purple house with green roof and orange lit windows...，每一条都准确描述了我喂进去的参考图。名字对不上，说明它理解错了，那就别往下走了。

坑一：Tripo的GLB比例是归一化的

这个坑极具迷惑性，因为它长得好像比例问题。

每一个Tripo导出的GLB都被归一化到最长边约等于1.0单位。下面这组是实测的高度值，注意它们并不都是1.0——因为撑满1.0的那条边未必是高：

```
eiffel-tower 1.0000 / haussmann 0.9096 / carousel 0.9973 / teacup 0.5037
bench 0.5491 / walker-rigged 1.0000 / tree-green 1.0000
```

解析包围盒能看清这件事：铁塔是 $0.686 \times 1.0 \times 0.687$ （高撑满），长椅是 $1.0 \times 0.549 \times 0.413$ （长撑满），出租车是 $0.522 \times 0.570 \times 1.0$ （深撑满）。引擎里 `scale = 1` 直接摆，等于把每栋楼当成一米大。表现出来是「东西都对，但看着很怪、很空、像个模型展览」。你会去调光照、调材质、调后处理，怎么调都不对，因为根因是比例。

正确做法四步：实例化后先量包围盒；`s = targetHeight / bounds.size.y`；摆位时把底面压到地面 `pos.y -= bounds.min.y * s`；宽扁件（车、长椅）按最长边归一，不按高度。

注意

walker-rigged 那一行是重新量过的，别拿包围盒去量绑骨件。我先前填的是1.0276，那是 `Box3.setFromObject` 的读数——而 §06 和 §12 都讲过，这个函数量 `SkinnedMesh` 会被当前姿势和那个不自动更新的包围盒缓存带偏。一个「归一化到1.0」的模型量出1.0276，超出的那一点本身就是量法不可信的信号，而不是这件资产的例外。改用 §12 那条正解重量了一次——直接读 GLB 里 `POSITION` accessor 的 `min` 和 `max`——这件的包围盒是 $0.9132 \times 1.0000 \times 0.1847$ ，撑满1.0的是高那一条，和其余六件服从的是同一条归一化规则。上面那四步里的「量包围盒」，对静态件够用，对绑骨件一律换成按 accessor 读。

验收信号很好记：给58米算出来的缩放是 $\times 58$ ，说明原始就是1.0，你数对了。

坑二：glTFast默认不带meshopt解码器

症状会把你带到完全错误的方向。GLB文件明明在磁盘上、`.meta` 也生成了，但 `AssetDatabase.LoadAssetAtPath<GameObject>()` 返回 `null`，日志只说「找不到资产」。

真相藏在 `.meta` 里，那是唯一给线索的地方：

```
reportItems:
- type: 0
  code: 20
  messages:
  - EXT_meshopt_compression
```

能正常导入的模型，这一项是 `reportItems: []`。

根因是 Tripo 的 CDN 上 `tripo_pbr_model_<uuid>_meshopt.glb` 这个变体带 `EXT_meshopt_compression`，而我的工程里没有 meshopt 解码器，`ScriptedImporter` 直接判导入失败。说清楚一点：**glTFast 本身是支持这个扩展的**——官方功能表上写着「 via package」，4.4.0 起装 `com.unity.meshopt.decompress` 就会定义 `MESHOPT` 宏、走解码路径。所以准确的说法是「默认不带解码器」，不是「解不开」。装那个包和本地解压二选一，我选了后者，理由在下面。

修法是在本地解掉再喂 Unity：

```
npx --yes @gltf-transform/cli copy in.glb out.glb
```

`copy` 会走一遍完整的读写流程，出来的文件只留下 `KHR_mesh_quantization`，Unity 那边直接就能导入。我选这条而不是装解码包，是因为它把问题挡在工程外面：22 个 GLB 一次处理干净，谁拉这个工程都不用先装对一个包。要是你的管线天天从 Tripo 拉新模型，装 `com.unity.meshopt.decompress` 更省事。

注意

别指望换 URL 绕过去。把地址里的 `_meshopt` 去掉换非压缩变体会返回 403，CDN 的签名是绑定到具体资源路径的。

坑三：批处理里新GLB要跑两遍

-executeMethod 跑的时候，新拷进来的GLB还没被导入。在方法开头加 `AssetDatabase.Refresh()` 不够，glTFast的导入是延迟的，而批处理里没有帧循环等它完成。

跑两遍就好：第一遍触发导入，第二遍资产就在了。我的构建脚本里这条是写死的。

坑四：把Light加在root上，整个物体会被挪走

```
var root = new GameObject("FerrisWheel");
root.transform.position = c; // 摆到 (-42, 0, 66)
var rimLight = root.AddComponent<Light>(); // ← 灯加在了 root 自己身上
rimLight.transform.localPosition = new Vector3(0f, hubY, 0f);
//          ^^^^^^^^^^^^^^^^^^ 这就是 root.transform, 整个轮子被挪到 (0, 17.5, 0)
```

这个bug最难查的地方在于它不报错。场景里有一个完整的摩天轮，只是它不在你以为的地方。我一度以为是相机机位写错了。

诊断方法：不要猜，直接打印。

```
var fw = GameObject.Find("FerrisWheel");
var wb = new Bounds(fw.transform.position, Vector3.zero);
foreach (var r in fw.GetComponentsInChildren<Renderer>())
    wb.Encapsulate(r.bounds);
Debug.Log($"root={fw.transform.position} "
    + $"包围盒 center={wb.center} size={wb.size}");
```

打印出来 `root=(0.00, 17.50, 0.00)`，一眼定位。灯要挂在子物体上。



摩天轮：圈、辐条、转轴是代码算的正圆，吊舱是Tripo生成的。这张是它被挪回原位之后的样子。

坑五：圆柱体的轴是Y，转90度的账要算对

Unity的 `PrimitiveType.Cylinder` 轴向是Y。绕Z转 ϕ 之后轴指向 $(-\sin\phi, \cos\phi, 0)$ 。摩天轮的辐条要沿半径，所以 $\phi = \theta + 90^\circ$ ；外圈要沿切线，所以 $\phi = \theta$ 。

我把这两个的90度弄反了，整只轮子炸成一把乱插的棍子。这是个看着像玄学、其实是三角函数的典型，算一遍就清楚了，别靠试。

坑六：前向渲染下，一个物体吃得下几盏像素光是有上限的

一座 140×150 米的园区，地面铺成一整块的时候，这块地面是一个物体。Unity只会挑几盏灯去逐像素照它，58盏路灯里绝大多数对地面毫无贡献。表现是「灯全开了却还是一片黑」。

注意

「几盏」具体是几，别记成一个写死的4。Built-in前向渲染下这个上限由

`QualitySettings.pixelLightCount` 决定，它跟着画质档走，不是引擎写死的常数——我这个工程里那一档的默认值是4，构建脚本里的注释就是这么记的。而把它调大并不能把这个坑填上。我最后在脚本里写的是 `QualitySettings.pixelLightCount = 8;`，58盏路灯照样有50盏进不来，地面的光池还是得画进贴图里。所以这个坑的性质不是「数字设小了」，是「实时逐像素光本来就不是用来铺满一座园区的」，往上调只会换来更慢的帧率。正解在下面那张表。

做法	结果
一整块地面+调亮灯	只有 <code>pixelLightCount</code> 那几盏生效，怎么调都不够

做法	结果
地面切成8×8网格	每块各自挑灯，边界出现光照接缝
把光池画进地面贴图	无缝、不限盏数、零开销

正解的道理是：灯的位置本来就是我自己算出来的，那就不该让引擎去实时算。共享一份 `LampPositions()`，地面贴图按它画光池、实际摆灯也用同一份，两者永远对得上。实体点光只负责照亮树和设施这些立在地上的东西。

光池衰减用 $1/(1+(d/r)^2)$ ，叠加时取 `max` 而不是相加（相加会在灯密处糊成白团）。光池半径 `R=13.5m` 是试出来的。和它配套的那几个后处理参数，请按字段名对，别按数字记：最后落在构建脚本里的是Bloom的 `intensity 0.38`、`threshold 0.92`，以及ACES之后的 `postExposure 0.25`（少数几个导演机位会把它单独抬到0.65到1.15）。这几个都是Post Processing Stack v2的字段，URP的Volume里名字和取值范围都是另一套。



广场看旋转木马。地面上那些暖色光池全部来自贴图，实体点光只负责照亮立起来的东西。

坑七：Bloom的阈值才是那个开关

`threshold = 0.72` 意味着比0.72亮的东西全都发光，连被照亮的墙面也一起炸，整个画面糊成一片白。夜景里只有灯泡和灯带该发光。

阈值给到 `0.92`，只让真正的高光炸开。这一条比`intensity`重要得多。（同样提醒一句：`threshold` 是PPSv2里Bloom的字段，URP的Volume版Bloom虽然也有一个阈值，但它和整条曝光链的关系不一样，0.92这个数搬过去不保证还是同一条界线。）



环绕机位。摩天轮那一圈灯泡和旋转木马檐口的灯带在发光，而被照亮的地面、树冠和长椅没有跟着一起炸。阈值0.92就是这条界线；给0.72的话整张图会糊成一片白。画面左边那条船是海盗船，右上角远处是鬼屋。

坑八到坑十：三个小的

中文文件名在macOS上会被NFD归一化。我写过一个「渲染成中文名再复制成ASCII名」的自校验脚本，结果排序时旧图把新图覆盖回去了（03-高处俯瞰.png 的Unicode码位排在 03-俯瞰全园.png 之后），白看了两轮旧截图。修法是渲染直接出ASCII文件名。

静态渲染里 **Update** 不跑。编辑器模式渲染时 `MonoBehaviour.Update/LateUpdate` 不会执行，所以旋转木马和摩天轮在静帧里是静止的、走路角色是T-pose。这些都是正常的，不是bug。要出带动作的帧得进Play模式。

glTFast导入的动画clip是子资源，播放不需要AnimatorController。它会把 `preset:biped:walk` 作为 `AnimationClip` 子资源挂进来，`AssetDatabase.LoadAllAssetsAtPath` 能枚举到。用Playable直接推给Animator：

```
using UnityEngine.Playables;    // ← PlayableGraph
using UnityEngine.Animations;   // ← AnimationClipPlayable / AnimationPlayableOutput

_graph = PlayableGraph.Create("Walker");
var playable = AnimationClipPlayable.Create(_graph, clip);
var output = AnimationPlayableOutput.Create(_graph, "anim", animator);
output.SetSourcePlayable(playable);
_graph.Play();
```

三个必须知道的点。第一，**using** 是两条不是一条，而且这三个类型分在两个命名空间里：`PlayableGraph` 在 `UnityEngine.Playables`，`AnimationClipPlayable` 和 `AnimationPlayableOutput` 在

UnityEngine.Animations。只加后面那一条， PlayableGraph.Create 这一行就是CS0246找不到类型，而报错指的是行、不是命名空间，很容易以为是包没装。第二， PlayableGraph 字段要标 [System.NonSerialized]，它含指针，不能让Unity序列化。第三，自定义的数据类要标 [System.Serializable]，否则存进场景后构建出来是空的。

那个最坏的失败：构建成功，启动即崩

构建出来的macOS应用启动就崩，日志只有一句：

```
The file '.../Data/level0' is corrupted! Remove it and launch unity again!
```

level0 是构建产物里的序列化场景文件，三个构建的 level0 都约1.0 MB、没有截断。

根因：场景文件里被写进了内联MonoScript。 Assets/Park.unity 里41个 m_Script 引用中，有27个是没有guid的纯fileID形式，指向场景文件内部的8个 !u!115 MonoScript 块：

```
--- !u!115 &140919112 ---
MonoScript:
  m_Name:                               ← 空
  m_ClassName: ThirdPersonCamera
  m_AssemblyName: Assembly-CSharp
```

正常Unity场景永远不该内联MonoScript，它只应存在于 .cs 资源的导入产物里。 m_Name 为空是「脚本对象已创建但资产注册未完成」的特征。触发条件是批处理里「编译脚本 → AddComponent → SaveScene」跑在同一个进程里，而中招哪些类是竞态：同一份代码，12个 Spin 安然无恙，另外8个类全中招，两次构建的中招集合还不一样。

注意

我在这上面栽得最重的一跤，是把「假正常」当成了对照。崩溃之后我做了个对照：不建主角，app能正常启动，于是判定问题出在主角那一套。独立审查者去翻那个对照app的运行日志，里面有**74行** The referenced script (Unknown) on this Behaviour is missing!。74=37个组件 × 2次运行。那个「正常」的对照版，39个自定义组件里有37个在运行时全部丢失，木马不转、摩天轮不转、行人不动，它只是窗口弹出来了而已。

验证一个构建产物，「进程还活着」是最弱的一级证据。正确的判据是三条同时看：运行日志里的装配和警告计数、应用自己截的图、组件数量对账。

解法是把场景里的自定义脚本清零：

层	内容
场景	只有纯数据：模型实例、灯、相机、CharacterController、BoxCollider、后处理组件，全是原生或包组件

层	内容
清单	把「谁该有什么行为、参数多少」写进 <code>Assets/Resources/park-manifest.json</code>
运行时	<code>ParkBootstrap</code> 用 <code>[RuntimeInitializeOnLoadMethod]</code> 读清单，按层级路径找对象、挂行为

场景文件里没有任何自定义脚本引用，就没有可损坏的引用。清单存的是层级路径不是名字（同级会重名），而且必须在重挂父级之后才记录路径，否则运行时找不到。

构建链上还加了一道闸门 `ParkVerify.SceneIsClean()`，检查场景里有没有无guid的 `m_Script` 或内联 `!u!115`，命中就报错并中止构建。宁可构建失败，也不要产出一个「构建成功、启动即崩」的app。



原生可玩版跑起来之后应用自己截的图（`-autoshot` 加 `-autowalk`，从应用内部调

`ScreenCapture.CaptureScreenshot`），不是窗口截图。日志同时报 运行时装配完成：37 个行为挂上，0 个对象缺失。

三条证据一起看，才算验收过了。

角色发白：根因不在灯，在角色

这条我判错过一次，值得记。

网页版和巡游片里所有行人和主角都是发白的白色人形，肉眼一看就是坏的。我第一轮的处理是调灯（路灯5.2降到2.1、曝光0.38降到0.26），然后判定「站在灯正下方仍偏亮属于可接受的夜景表现」。

复核推翻了这个判断。根因是角色本身：Tripo出的这个角色是白T恤加浅灰裤，基色贴图均值0.58，而长椅是0.18、树是0.44。同一套灯光下它比别的东西亮三倍，再过一遍曝光、ACES和Bloom就成了白影。

修法不动灯光，把角色材质克隆一份存成独立材质，基色乘0.5、粗糙度拉满、金属度归零。



角色材质压暗之后。光池外的行人已经能看出上衣、裤子和头发的分层，不再是一团白影；站在灯正下方那两个仍然偏亮，源记录里也保留了这条限定。2026-09-11复核后重建。

注意

别拿编辑器渲染判角色。编辑期离屏渲染里角色是T-pose，而且入口机位正对两排路灯光池叠加处，躯干有55%像素削顶，看着像没修好。真正的判据是可玩版运行时抓帧。

WebGL体积：165 MB压到51 MB

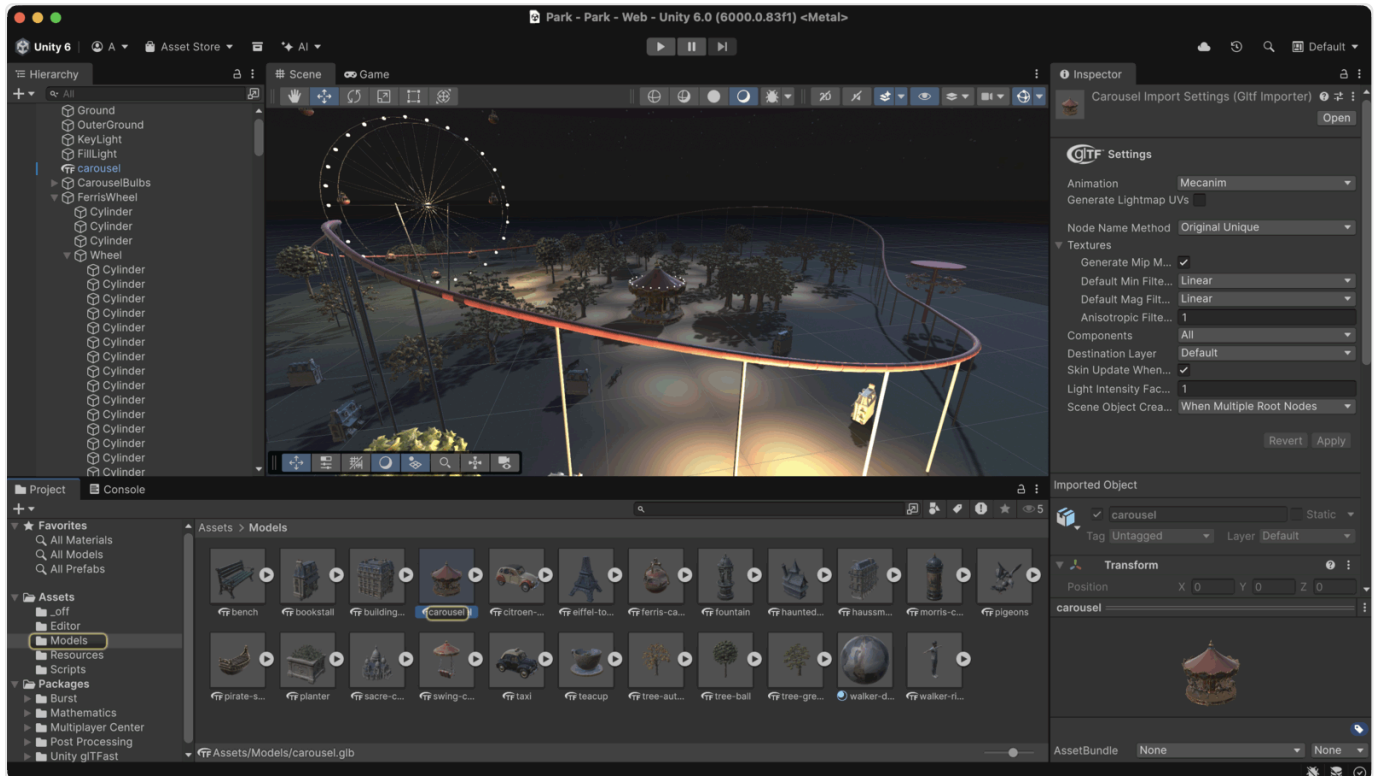
第一版WebGL构建165 MB，其中 `WebGL.data` 一个文件就152 MB。源GLB一共才58 MB，构建出来却是152 MB，还是brotli压过的。

量了贴图才明白，而且这笔账要按张数数，不能按件数数。六座游乐设施每件挂三张：基础色是 8192×8192 ，法线和ORM各是 4096×4096 ，一件合计1.007亿像素；六件就是18张、6.04亿像素。加上园区里另外十六件（十五件是三张 1024^2 ，只有 `citroen-2cv` 那件是三张 4096^2 ），22个GLB的贴图总量是7.01亿像素。一张 8192^2 在Unity里按RGBA32展开是268 MB，七亿像素全摊开是2806 MB，带上整条mipmap链（ $\times 4/3$ ，口径和 § 12 那笔账一样）约3.7 GB。源文件里这些图是JPEG，所以在磁盘上不占地方，进了构建就全摊开了；brotli回压到152 MB，量级对得上——这是一个量级判断，不是逐字节对账。

贴图这件事要按模式说，别记成一句「Tripo出8K」。这个园区用的是v3.1这条路，出的是base color、normal、ORM三张，所以下面那几条 `resize` 命令才需要按槽位分尺寸。P2.0那条路只出一张base color，没有normal也没有ORM，那边的处理方式完全不同，在 § 12。两条路的贴图工序不能互相照抄。

注意

这条对任何用glTFast的项目都成立，不是本项目的特例：glTFast的编辑器导入直接创建 **Texture2D**，不走 **TextureImporter**，所以Unity那一整套平台贴图压缩设置从来没被应用过，构建里存的是未压缩的RGBA32加mipmap。glTFast的 ImportSettings 里只暴露了mipmap和过滤模式，没有贴图尺寸和压缩的开关。



选中 `carousel.glb` 后的Inspector，顶上写的是Gltf Importer而不是Unity内建的Model Importer。Gltf Settings 那一栏能看到的全部选项：Animation方式、Generate Lightmap UVs、Node Name Method、Textures、Components。没有任何一项管贴图尺寸或压缩格式，这就是上面那条结论的现场证据。

引擎侧没有开关，就在源文件上处理。贴图名字带槽位标识，可以分开定尺寸：

```
R="npx @gltf-transform/cli resize"
$R --width 512 --height 512 --pattern "*ormal*" in.glb t1.glb
$R --width 256 --height 256 --pattern "*ORM_*" t1.glb t2.glb
$R --width 256 --height 256 --pattern "*_rm" t2.glb t3.glb
$R --width 1024 --height 1024 t3.glb out.glb
```

分配逻辑：基础色1024（决定观感，给足）、法线512、金属粗糙度256。基础色1024对1080p输出是3倍过采样，木马在画面里最宽也就640像素。

这四行有两个坑。--pattern 是glob而且大小写不敏感，`*ORM*` 会把 `NormalGL_...` 一起命中（「Normal」里有「orm」），法线也被压到256；改成 `*ORM_*`（带下划线）才分得开。`resize` 只会缩小、不会放大，所以必须先做小尺寸的pass、最后做兜底的大尺寸pass，顺序反了没法补救。

	优化前	优化后
WebGL总大小	165 MB	51 MB
WebGL.data	152 MB	48 MB
模型源文件	58 MB	20 MB
贴图总像素（22件×3张）	701.5 M	31.3 M
原生Mac应用	2.6 GB	191 MB

「贴图总像素」那一行是从22个GLB里逐张读图像头加出来的，66张一张不落，不是估的。降完之后是31.3 M而不是 $22 \times 1.376 \text{ M} = 30.3 \text{ M}$ ，多出来的那1 M有个具体出处：haussmann那件的ORM贴图名字既不含 `ORM_` 也不以 `_rm` 结尾，两条 `--pattern` 都没命中它，最后落在兜底那一档被定成 1024^2 而不是 256^2 。按名字分槽位就得接受这个：名字不守规矩的那几件会漏网，而它不报错。

原始8K模型单独备份着，要出更高规格的版本可以回到那一份。



用CDP驱动headless Chrome抓到的实际渲染帧。这张图是为了破一个假阴性：我查 `window.unityInstance` 得到 `undefined`，一度以为运行时没起来，其实Unity默认的加载模板根本不往 `window` 上挂这个变量。判据必须是画面，不是某个变量。

出片管线：把已经在跑的东西录下来

园区里的东西本来就在动。静帧传达不了这一点，所以出片不是「加个渲染」，而是把已经在跑的东西录下来。三个关键决定。

用 `Time.captureFramerate` 定帧，不用实时录屏。它让游戏时间按固定步长走、与真实帧率脱钩，每一帧都是确定的。实时录屏会因为掉帧而抖动，而这台机器的帧率并不稳定。

渲到RenderTexture再读像素，不走 **ScreenCapture**。后者只能截屏幕分辨率（这台机器是3024×1898），出不了标准1920×1080。

存JPEG不存PNG。1080p一帧PNG约2 MB且编码慢一个数量级，JPEG(q93)约300 KB。一千五百多帧，PNG要3 GB，JPEG只要450 MB。（定稿55秒×30 fps=1650帧；源记录里那条 `-frames 1515` 是中间版本留下的。）

```
open -a Build/Park-Play.app --args -logfile /tmp/vc.log \
    -capture /tmp/frames -cine -frames 1650

ffmpeg -y -framerate 30 -i /tmp/frames/f%04d.jpg \
    -c:v libx264 -preset slow -crf 17 \
    -pix_fmt yuv420p -movflags +faststart out.mp4
```



巡游片第一镜，入口大道推进。11个航点、55.0秒，镜头之间用SmoothStep缓动（ $u^2(3-2u)$ ），匀速位移会很机械。



巡游片收尾的俯瞰拉远。九张关键帧对应成片5.00到54.00秒。

镜头编排连错三次，三个不同的盲区

同一件事我错了三次，每次错法都不同，所以记下来。

错法一：分镜里漏了资产。第一版10个镜头里没有海盗船，因为列机位的时候它还不存在，我凭记忆列，记忆里就没有那个还没做出来的东西。列分镜要把所有资产过一遍。

错法二：镜头路径穿过实体。把海盗船排在摩天轮之后，两者相隔46米，而镜头位置是线性插值的，它不会绕开几何体。结果那5秒里镜头从船体正中穿了过去，画面全是木板。「两点之间直线最短」在镜头编排里是缺点不是优点。

错法三：看向点跟着位移一起插值，镜头全程在甩头。这是最难发现的一个。

```
float er = SmoothStep(Mathf.Clamp01(u * 1.9f)); // 先转头
transform.position = Vector3.Lerp(a.pos, b.pos, e); // 再位移
Vector3 lookAt = Vector3.Lerp(a.look, b.look, er);
transform.rotation = Quaternion.LookRotation(
    lookAt - transform.position, Vector3.up);
```

看向点要跑得比位置快，而且朝向要每一帧从当前位置朝当前看向点重新解算。先转过去对准主体、再推进，这才是运镜；插值两个四元数只是甩头。

还有一脚：应用失焦后暂停，录到第194帧卡死了63分钟。Unity应用默认 `runInBackground = false`，一失焦整个应用就暂停。这和浏览器里后台标签页 `rAF` 被暂停是同一类问题：宿主认为你不在前台，就停掉你的渲染循环。两层都要修：应用内设 `Application.runInBackground = true`，外面用 `caffeinate -dims` 防App Nap。

核心建议

卡死那63分钟我完全不知道。第一版监视器是「每75秒报一次帧数」，能看见进度，但也意味着它不会告诉我出事了。改成只在两种情况下出声：渲染完成，或者连续5分钟没有新帧。**沉默即正常**。不区分「在跑」和「卡住」的监视器会退化成噪音，跟没有一样。



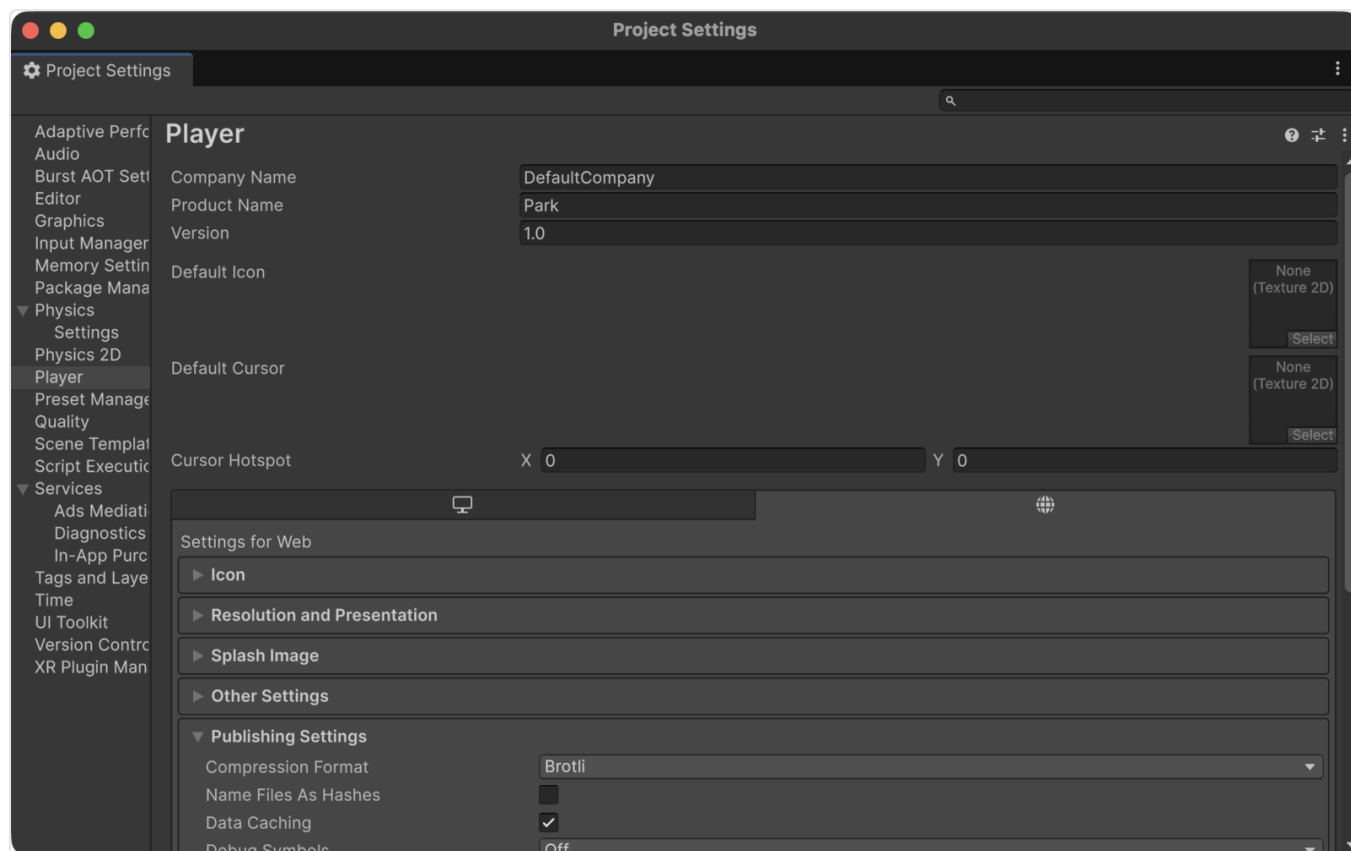
俯瞰全园。六座设施加过山车轨道加行人路网，Tripo负责形体、代码负责机构，这张图上两者各占一半。



巡游片里的一帧。成片55秒/ 1920×1080 / 30 fps / h264+AAC，音轨全部由ffmpeg现场合成（布朗噪声过低通做环境床、粉噪声过带通做风声、两个正弦做和声垫），没有任何外部音频素材，因为独立审校指出素材库那支BGM的授权范围无法确认。

发布：WebGL目录扔上去就行

构建用的是Brotli加 `decompressionFallback`，加载器自带JS解压，不要求托管方为压缩文件配 `Content-Encoding` 头，Vercel、Cloudflare Pages、GitHub Pages都能直接扔整个目录。



Project Settings → Player → Settings for Web → Publishing Settings，Compression Format选的是**Brotli**。同一栏里还有Name Files As Hashes、Data Caching、Debug Symbols。这个面板在Unity 6里叫「Settings for Web」，不叫WebGL。

产物文件后缀是 `.unityweb` 而不是 `.br`。判断它到底压没压，看文件头：

```
xxd -l 40 Build/WebGL/Build/WebGL.data.unityweb
# 00000000: 6b8d 0055 6e69 7479 5765 6220 436f 6d70  k..UnityWeb Comp
# 00000010: 7265 7373 6564 2043 6f6e 7465 6e74 2028  ressed Content (
# 00000020: 6272 6f74 6c69)...          brotli)
```

注意要取40字节，`head -c 32` 只会截到左括号为止，看不到 `brotli`。

三个交付物用的是同一套资产，但不是同一份文件。网页版的贴图为了体积做过降采样（基础色1024 /法线512 /金属粗糙度256），看着和片子几乎没差别，严格说不是同一份资源。别在片子里说「和网页版一模一样」。

§15 谁负责哪一层

Who Owns Which Layer — A Relay, Not a Race

一条能让你当场决定「这件事交给谁」的分工线：资产交给3D生成，规划和组装交给coding agent。再加一条只有一句话、但我整本书都在用的取舍判据，和一份诚实到写了「没测」的分层诊断。

那个问题本身问错了

做这本书的过程里，我被问得最多的一句是「Tripo和Blender谁强」。偶尔换个版本：「有了AI生成还要不要学Three.js」。

这个问题有个陷阱。它们在流水线的不同环节上，压根不在一个赛道。问谁强，就像问钻头和螺丝刀谁强。但我把巴黎、游乐园、小王子这三条线全部做完之后，发现连「几个工具之间怎么排」都不是最该先问的那一层。真正把人卡住的分界在更前面：一边是资产，一边是把资产变成一个能玩的东西。这两件事的性质完全不同，需要的能力也完全不同。

接力：资产只是一半

我在片子最后说的那句判断，是这一整节的地基：

资产只是一半。真正把这些拼成一个能玩的游戏的，是前面那个coding agent。它负责规划、写代码、把东西拼起来。这两个是接力的关系。



	第一棒 · 资产	第二棒 · 规划与组装
谁在跑	3D生成工具（我用的是Tripo）	coding agent（我用的是Codex）
干的活	一张参考图进去，一件能进引擎的模型出来；要能动的，在这一棒里就把骨绑好、动作挑好	拆需求、定架构、写代码、把一堆文件拼成一个能开能玩的东西，再自己写验收脚本回头测它
交出什么	文件。文件不是作品	那个能玩的东西本身
缺了它会怎样	第二棒做出来是一堆方块	资产躺在硬盘上，什么都不是

不是替代关系，是接力关系。接力棒从左往右传，两棒解决的问题完全不一样，谁也替不了谁。

「只是一半」这四个字不是客套。巴黎那23件GLB躺在硬盘上的时候，它们什么都不是：没有比例、没有碰撞、没有摆放规则、没有可视距离分级，甚至没有一个统一的朝向。把它们变成一座能开着车穿过去的城，是另一件完全独立的活，而那件活是coding agent干的。

反过来也一样。第二棒再强，手上没有像样的资产，做出来的就是一堆方块。我那座城在换掉行道树之前，抬头看到的是一排深绿色的积木（§ 17那两张对比图），代码再怎么写也长不出树皮和叶片的层次。

核心建议

判断一件活卡在哪一棒，有个特别快的办法：问「现在缺的是一个文件，还是一条规则」。缺文件是第一棒的事，缺规则是第二棒的事。缺的是「这东西该摆在哪、按什么密度摆、多远之后换成简化版」，那再生成十件模型也没用。

这本书的编排其实就是照这个顺序来的：§ 01到§ 14在讲怎么跑好第一棒，§ 16到§ 19在讲第二棒怎么把接力棒接住。这一节是中间那个交接区。

你会开到跟前细看的，才值得生成

知道了有两棒，下一个问题马上就来：具体到场景里的每一个东西，它该走第一棒（生成一件资产），还是根本不用走，直接让第二棒用代码画出来？

这条判据是我做完整座城之后能给出的、唯一一条可以照抄的工程判据：

你开车会开到跟前细看的東西，用Tripo做；你只会远远瞥一眼的，代码画就够了。这样既保住了近景的细节，也不会让帧率掉下来。

它好用是因为它把一个技术问题翻译成了一个玩家视角的问题。你不需要先懂三角面预算、不需要先懂draw call，你只需要问：**玩家的臉会不会湊到这个東西前面去。**

巴黎那座城真实的归属是这么分的，我在片子里也是这么说的：

推荐

Tripo生成的

铁塔、凯旋门、圣心堂、车，还有一部分街边的建筑和街具。

→ 全是你会开到跟前、甚至会停下来绕一圈的东西

不推荐

代码画出来的

成片的街区、桥、塞纳河。圣母院、卢浮宫这几个地标也是代码。

→ 全是构成城市底色、但你只会远远扫一眼的东西

所以这不是一整座城都是AI建模出来的，真实情况是混搭的。这句话我在片子里专门留了一拍来说，这里也要留一段来说，因为它是这整套方法能不能被复制的前提。看一个漂亮的demo然后以为「它整个是AI生成的」，照着做一定会撞墙：你会为了铺满一座城去生成上千件模型，钱、时间、帧率三样一样都不够。

注意卢浮宫和圣母院这两个例外。它们是地标，按理说该生成，但玩家在我这条赛道上只会从河对岸掠过，所以它们被划到了「远远瞥一眼」那一侧。判据认的是玩家的实际路线，不是这个东西在现实里有多重要。



判据的正面样本：铁塔是你一定会开到脚下的东西，所以它是Tripo生成的GLB。它背后那一整片白色的城、河、桥，你不会停下来看，所以是代码画的。

这条判据的第二次兑现是在游乐园那条线上，而且分得更干净：木马、摩天轮的吊舱、飞椅、茶杯、鬼屋是Tripo生成的，转的圈、轨道、转轴这些结构件是代码写的。游客的脸会贴到木马上去看鬃毛，但没人会去数摩天轮那一圈钢架的接头（那一轮的完整分工表在 § 14）。

三件工具的真实位置

把接力棒的路径摊开，中间会经过的几件东西就各就各位了：



三者摊开成一张表就没什么可争的了。最后一行是我建议你真正记住的那一行，它写的是交接时你要自己付的账。

	Tripo	Blender	Three.js / Unity
本质	AI 3D资产生成器	专业DCC软件（人工操作）	渲染库/游戏引擎
输入	一张图/一段文字	空场景，从零手工建	代码+资产文件

	Tripo	Blender	Three.js / Unity
输出	3D资产（GLB / FBX / OBJ）	可编辑工程+渲染成片	可交互的页面或应用
在接力里的位置	第一棒的主力	第一棒的返修车间，按需才开	第二棒的落点
擅长	打形、量产、占位、铺量	精度、绑定、动画、hero资产	交互、零门槛分发
交接时要自己付的账	面数要压、贴图要降、朝向和比例要归一	导入器会塞辅助物体、命名要自己整理	资产得先备齐，它不生产资产

Blender在这张表里的位置值得多说一句。它不是必经的一站，而是**第一棒的返修车间**：需要绑定、需要动力学、需要面部的东西才拐进去，其余的直接往第二棒传。我巴黎那批资产一件都没进过Blender。

结合使用的三种真实模式

框架讲完，问题就变成「我这件活走哪条」。实际跑下来只有三条。

模式A：Tripo → Blender → 引擎

角色走这条。需要绑定、需要动力学、需要面部的东西，绕不开Blender那一棒。X上做角色的创作者普遍是这么干的：Tripo分别生成头发、头、身体，然后在Blender里合并、绑定、做摇摆物理，半天做完一个角色。

模式B：Tripo → 引擎直连，跳过Blender

这条是我自己验证的。巴黎那23件资产，建筑、道具、载具、行人、地标，生成完直接导GLB、压缩、进Three.js，**全程没有经过Blender**。

实证结论：对场景类资产，这条路完全走得通。我由此省掉了整个Blender环节。

边界条件也是实测出来的三条。**面数要压**，Tripo默认输出1.6到1.8万三角面，装进Web场景必须减面加压缩。**减面有下限**，我那两棵巴黎梧桐压到7473面就交付了，再往下分枝拓扑就崩（各类资产的减面下限见 § 04；这两棵是09-10那版，后来被P2.0的七叶树换掉，见 § 18）。**贴图档位影响近距离观感**，我选的1k档，凑到墙上看会糊。

模式C：AI生成+程序化混搭

这是最容易被忽略的一条，因为它不是「用不用AI」的二选一，而是上面那条判据在同一个场景里同时落两次。

巴黎全城近景行道树用Tripo生成的GLB替换，中景、远景和公园球冠保留程序化几何——**近景要细节，远景只需要「那儿有个绿色的东西」**，而AI模型的面数成本撑不住两万株的量。**替换前后的对照图和这笔面数账在 § 17，集成时踩的坑在 § 18。**

量化的账在 § 13：14栋高细节Tripo建筑的总代价是多17个draw call、多15%三角面，帧率完全没掉。**用15%的三角预算买下赛道沿线全部的高细节，其余程序化铺满。**

同一个场景里，代码和AI怎么分

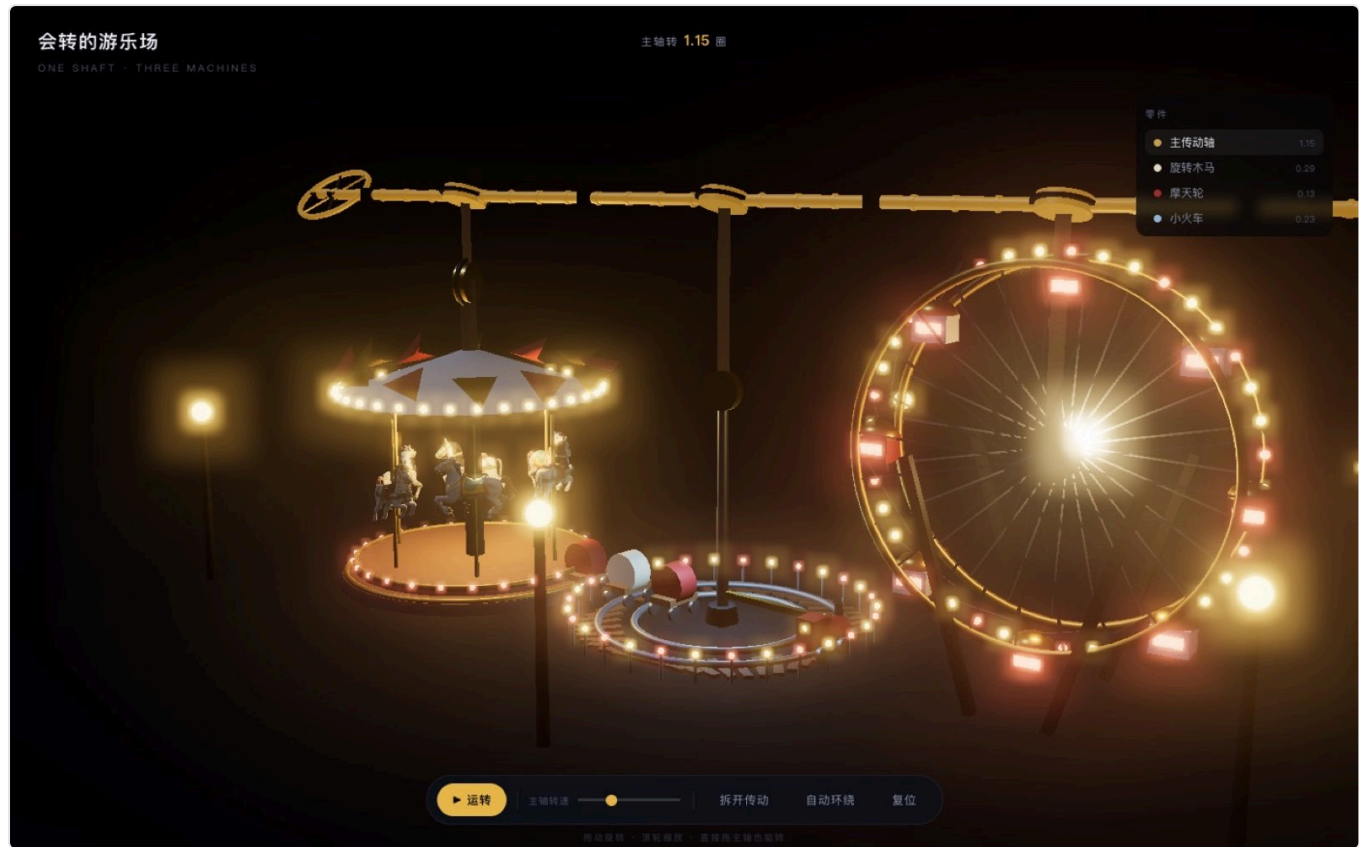
「跟前细看还是远远瞥一眼」管的是距离。还有一类东西距离很近、但照样不该生成，那就是**尺寸必须精确的**件。

四冲程发动机那轮给出了这条补充判据（那台机器本身、以及我花50积分生了一根曲轴又放弃的经过，在§20）。那台机器的零件来源是这么分的：

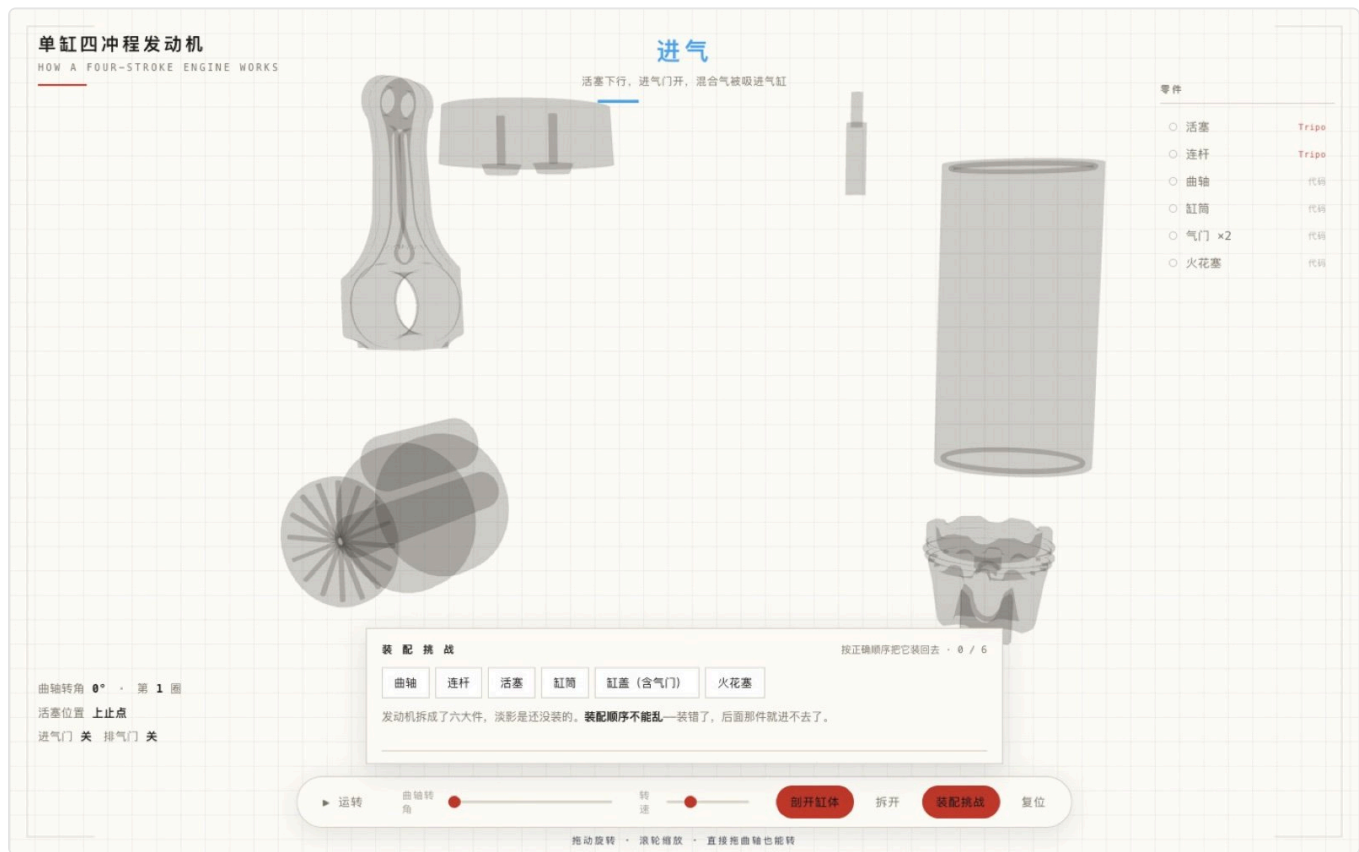
零件	来源	为什么
活塞	Tripo生成	形状复杂、有环槽和裙部细节，AI生成比手搓划算
连杆	Tripo生成	工字截面、大小头圆环，同上
曲轴	代码	曲柄销的位置必须精确 ，偏一点整台机器就是错的
缸筒/缸盖/气门/火花塞	代码	都是回转体，代码更干净，还能做半透明剖视

补充判据一句话：决定机构对错的尺寸件用代码，决定观感的形体件用AI。换个问法更好用：**这个件错了，别人一眼能看出「不对」吗？**能，就写代码。

两条判据合起来，覆盖面就完整了：**先问玩家会不会凑近看，再问它错了会不会被一眼看穿。**前一条管画面预算，后一条管正确性。巴黎那座城里的桥面高程、街廓碰撞、路网走向全部落在后一条上，所以它们哪怕就在车轮底下，也照样是代码算的。



《会转的游乐场》：一根主传动轴带旋转木马、摩天轮、小火车，传动比写死在代码里（木马0.25、小火车0.20、摩天轮0.115），面板上那几个数是各自已经转过的圈数。这一版里机构几乎全是代码，Tripo只出了木马上的马。所以它是「尺寸件用代码」这半条判据的样本，另外半条看下面那台发动机。



发动机的装配挑战：六步顺序，点错了给的是真实的机械约束（「缸筒一罩下来，活塞就没法从上面压进去了」）。这条交互之所以成立，前提正是曲轴、缸筒这些尺寸件是代码算的，位置精确到可以判对错。

七步变四步，省掉的是哪几步

第一棒内部还能再拆细一点。行业里讲这件事常用的拆法是：传统流程里一个角色从概念到进引擎要走七步，AI流程是四步。（下面这张表是行业通用口径，不是我自己的实测；我没有请一位熟练模型师来跑对照组。）

	传统流程（熟练模型师）	AI流程
步骤	概念图 → 三视图 → 高模雕刻 → 低模拓扑 → 展UV → 贴图烘焙 → 引擎调整（7步）	概念图 → Tripo生成 → 直出PBR材质 → 导入引擎（4步）
一个角色耗时	三五天起步	很快出初稿，收尾看要求

被压掉的三步是**高模雕刻**、**低模拓扑**、**展UV**。这三步恰好是最需要手上功夫、也最消磨人的三步。

但注意「收尾看要求」这五个字。压掉三步不等于那三步的产出不再需要，只是**它们的质量由模型给你定了，你要么接受，要么回Blender自己收**。我在E3和E4里测到的碎片化、权重塌，本质上就是「低模拓扑」这一步被跳过之后留下的账。

独立评测普遍提到AI网格的几条技术特征，和我自己测出来的能对上：多边形分布均匀、不集中在细节密集区；边环不贴合解剖结构，做动画时容易挤压变形；手指、耳朵这类复杂区域有n-gon和捏缩顶点，UV有接缝；高精度档直接导进引擎会严重拖帧，必须再优化。这几条都有对应的绕法，分别在 § 04、§ 06、§ 12里。

最后那条我在 § 14量到了具体数字：六个8K贴图的资产让WebGL构建从51 MB膨胀到165 MB。这就是「**引擎调整**」那一步没有消失的证据。

接力棒掉在地上的三个地方

接力赛真正丢时间的地方从来不是某一棒跑得慢，是交接。这套流程里有三个交接面，每一个都有固定的成本。

交接面	要付的账	在哪一节
Tripo → 引擎	面数要压、贴图要降、比例要归一化、朝向要对齐、PBR金属要配环境贴图	§ 12 / § 14 / § 17
Tripo → Blender	碎片化的网格绑不出好权重、命名带uuid不好定位、导入器会塞进辅助物体	§ 13
引擎 → 发布	子路径托管要相对路径、沙盒可能禁用网络请求、压缩格式要确认真的压了、发布包别把归档目录一起带走	§ 12 / § 14 / § 19

这三张账单加起来，比任何一棒本身的耗时都长。知道它们存在，比知道哪个工具更强有用得多。

卡在哪一层：一份带「没测」的诊断

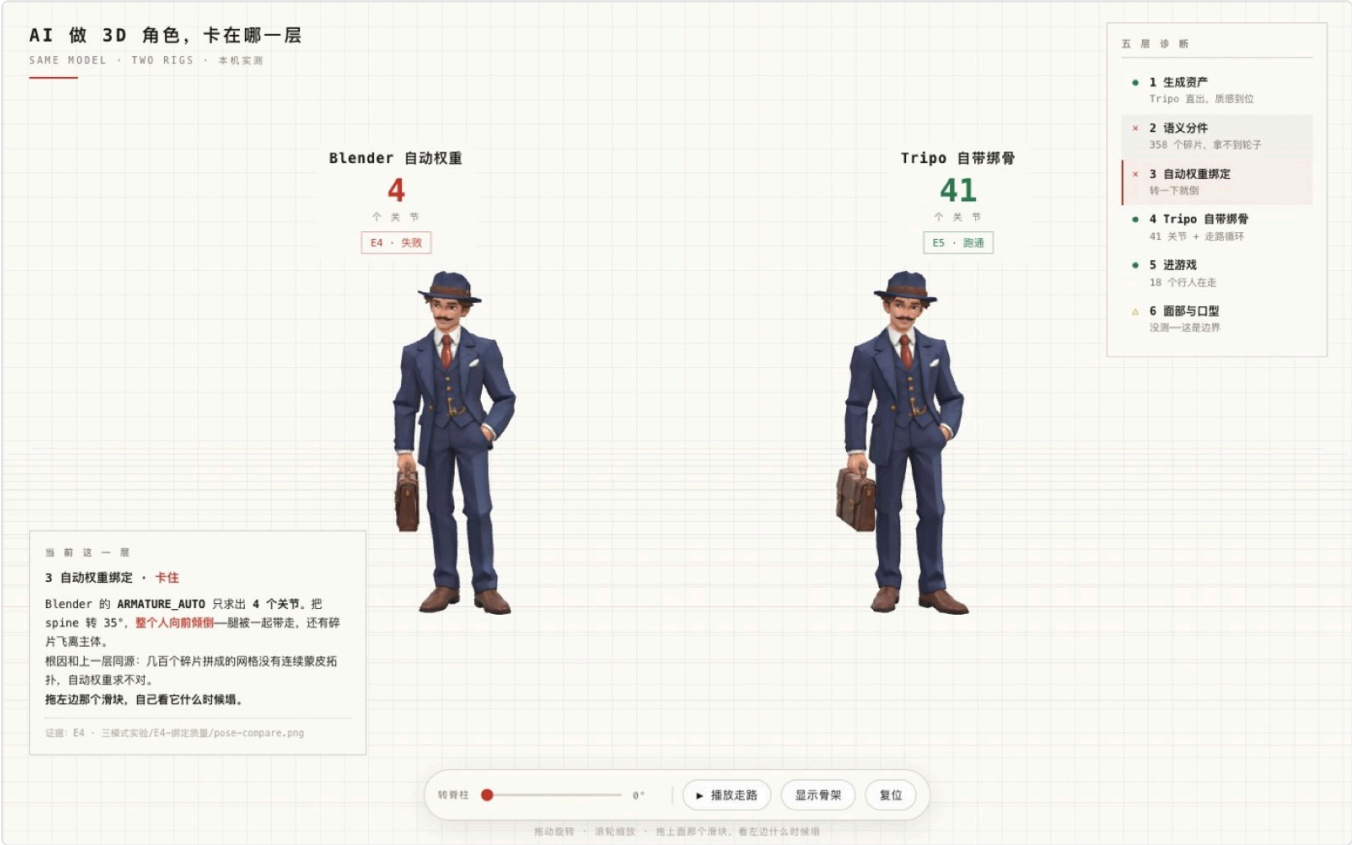
「AI能不能做3D角色」这个问题，答不出「能」或「不能」。我把它拆开一层一层判，判到哪算哪：**前五层有结论，第六层是边界。**

层	结论	证据
1 · 生成资产	✅ 成立	29个独立资产，16711到46196面，默认落在可用区间（这个29和城里那23件数的不是同一批，口径见 § 16）
2a · 几何连通分量分件	❌ 不成立	E3: Blender按连通分量（loose parts）把2CV切出358个碎片，拿不到轮子
2b · 语义分件	✅ 成立	§ 05: Tripo分部件生成的精细档拆出77个部件，轮子、车门、车顶各自独立
3 · 自动权重绑定	❌ 不成立	E4: 4关节，转一下脊柱整个人就塌
4 · Tripo自带绑骨	✅ 成立	E5: 41关节+ 2.375秒走路循环，20积分
5 · 进游戏	✅ 成立	巴黎demo里6个绑骨行人真的在走，另12个是五种静态行人站着（见 § 18），桌面加移动端全PASS
6 · 面部与口型	△没测	所以不下结论。这是我的边界

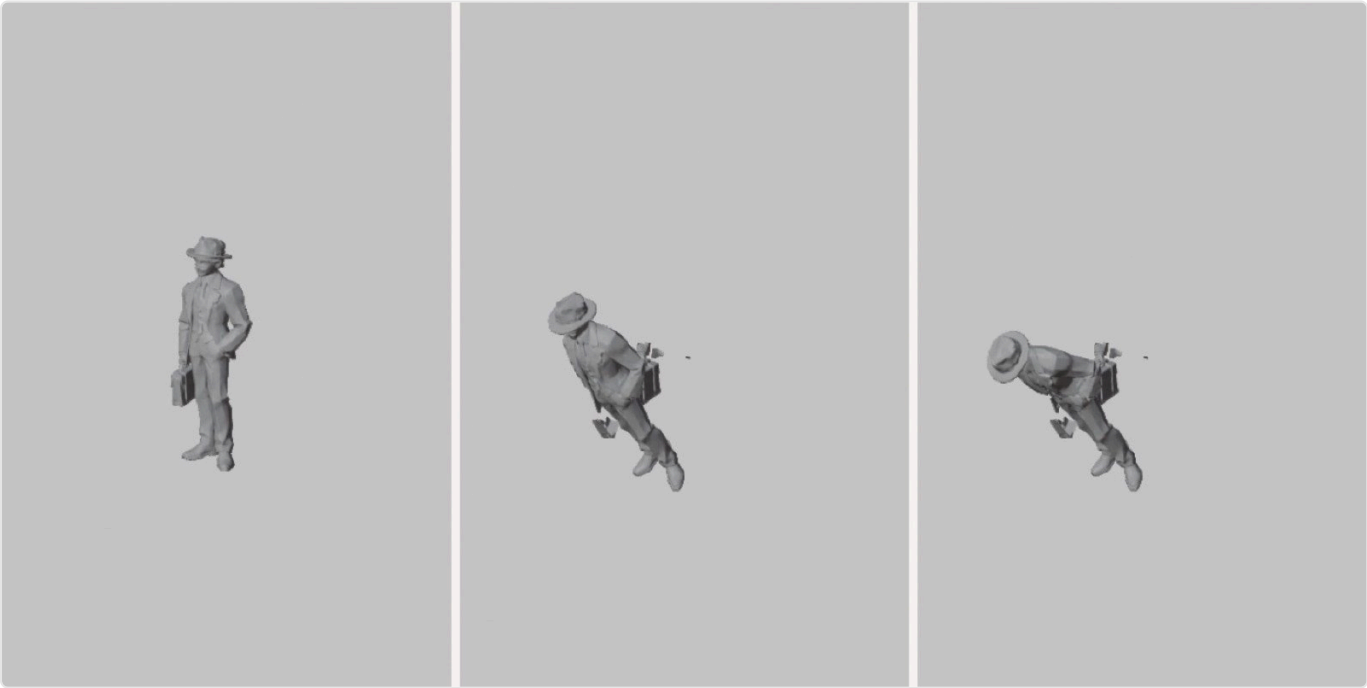
第2层拆成两行，是因为它在我这里出过两个相反的结果，而两个都对：**同样是一辆AI生成的车，用几何的刀切不出轮子，用语义的刀切得出来。**连通分量只认「这堆三角形彼此连着吗」，而AI生成的网格本来就是一层碎壳，轮胎和挡泥板之间根本没有几何缝；语义分件认的是「这块是什么」，所以它能在没有缝的地方划一刀。**这两行的差别，正是 § 05那一节存在的理由**——如果只看2a那一行就收工，你会得出「轮子转不了」这个错结论，而它只是「别用那把刀」。

核心建议

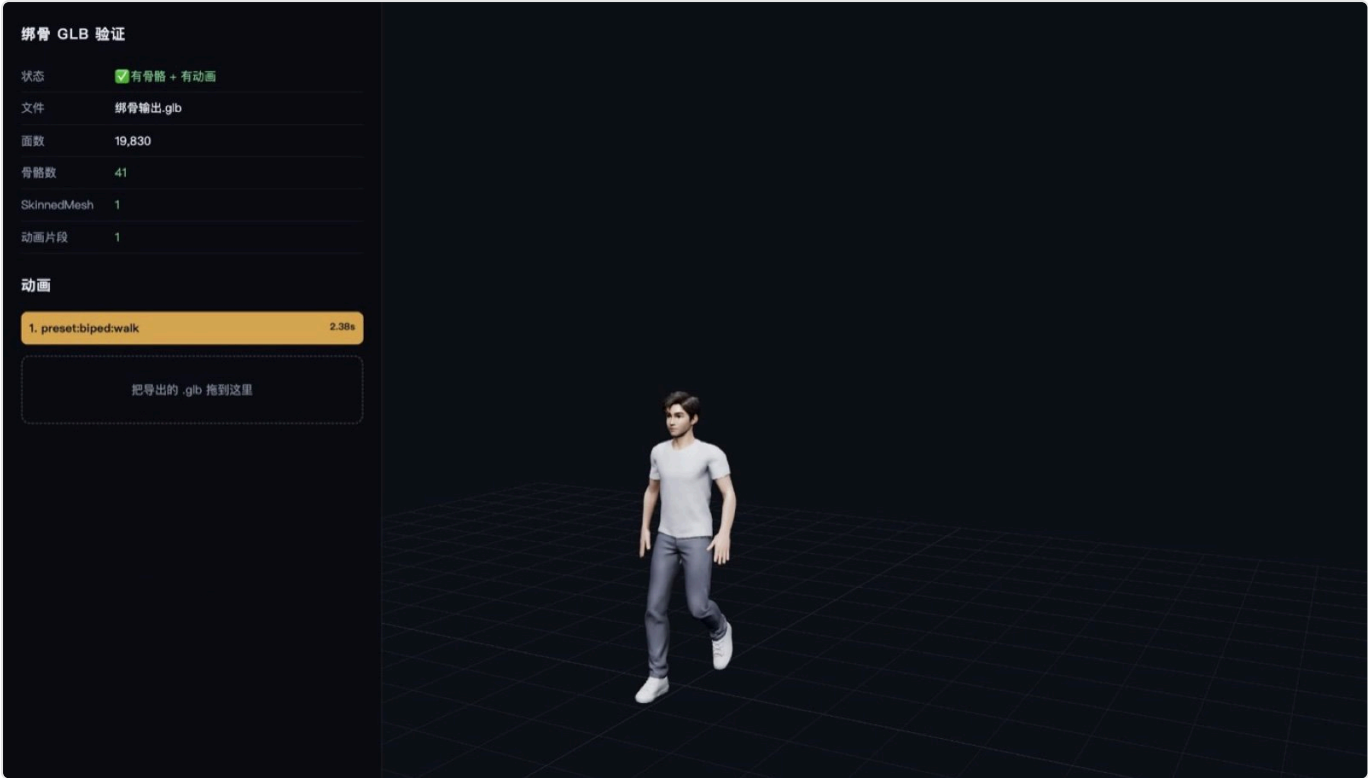
第6层是这份诊断的骨气所在。不吹也不黑，测到哪说到哪。一份全是对勾的诊断没有人会信，一份全是叉的诊断说明作者没真做过。



把第3层和第4层做成一个实时对照：同一个模型，两种绑定，一个共用的滑块（页面副标题写的就是SAME MODEL · TWO RIGS，左右确实是同一个戴帽提公文包的行人）。观众自己拖那个滑块，塌是他亲手拖出来的。点「显示骨架」模型会压成半透明，左边只有一根竖线，右边是一副完整骨架。



第3层的样子：Blender自动权重。左起A静止、B转 spine 35度、C再转 head 30度，公文包飞离主体、脚离地。原图顶部那条中文标签渲染时缺字体、是一排方块，进书前已裁掉。细节见 § 13。



第4层的样子：Tripo自动绑骨出来的行走循环。左侧面板的读数是41骨骼、1个SkinnedMesh、1段 preset:biped:walk、19,830面，动画拆开是123个通道、2.375秒一圈。这一件是 S 06里那个白T恤T-Pose男性，和上一张不是同一个角色，上一张是巴黎那个西装行人。别拿123去认角色：41关节的件通道数都是 $41 \times 3 = 123$ ，巴黎那个行人也是这个数，为什么有的件会变成126，S 06讲了。真正「同一模型两种绑定」的对照在前面那张方案D页面里。

关于「AI做不到的那一层」，一条我决定不当论据用的话

调研时我拿到一段转述，说VAST的首席科学家在一次访谈里划过一条边界：环境道具、场景组件、简单NPC、载具已经接近可直接使用；核心角色和复杂动画，尤其是面部和动力学绑定，仍是下一阶段最难解决的门槛。

这段话和我那六层诊断的结果吻合得有点好，正因为如此我更要说清楚它的来路。

注意

这是转述，出处目前只有「播客访谈」四个字，我没找到可以点开的一手来源。所以本书不把它当论据用，只当成一条「有人也这么看」的旁注。我那六层诊断的每一条都有自己的实验编号和产物文件，站不站得住只取决于那些实验本身，不取决于这段话。引二手材料之前先确认它还活着，引文准确不等于引用安全。

能自己测出来的，就不要靠别人的一句话。

那我到底该用哪个

把这一节压成四句话，贴在你工位上。

1 从零到有个东西 → Tripo

打形、铺量、占位、把脑子里的东西变成文件。这是第一棒。

2 从有个东西到是个精品 → Blender

但只有当你真的需要绑定、动力学、面部或hero级精修时才开它。只做减面和压缩的话，命令行更快（见 § 13的E0）。

3 从是精品到能玩 → Three.js或Unity

要在网页里、要零门槛分发、要人点开链接就能玩，选Three.js（§ 12）。要实时阴影、后处理、出片和原生可玩版，选Unity（§ 14）。

4 但上面三句里没有一句是「谁来做」 → coding agent

选哪个引擎、怎么分模块、资产怎么摆、性能怎么调、出了问题怎么定位，这些全是第二棒的活。你要做的是把需求和判据说清楚，具体写法在 § 18。

拿不准的时候，问自己这五句

上面四句是大方向。具体到手上这一件东西，我用的是这五个问题，按顺序问，问到哪停在哪。它们其实就是前面那两条判据的展开。

①它错了，别人一眼能看出「不对」吗？能，写代码。正圆、闭合轨道、精确的销距，这类件不能交给生成。

②镜头会怼到它脸上吗？会，那它值得生成，也值得预留重拓扑和人工精修的时间。只会远远瞥一眼，代码画就够了。

③它要不要动？不动就是场景资产，走模式B直连引擎。要动就得往下问第四条。

④动的是整体还是关节？整体位移、旋转、缩放，引擎代码就够了。关节要动，那就是绑骨，先试Tripo自带的（§ 06），别上来就开Blender。

⑤这个东西要出现几次？一次到几十次，生成；上千次上万次，程序化。中间地带就混搭，近景用生成的、远景用程序化的。

五个问题里没有一个问「哪个工具更强」。问题永远是「这件活属于哪一棒」，工具是答案的副产品。

还有一句是留给不会写代码的人的，也是这本书的前提：这两棒你都不需要亲手写代码。Tripo是点界面或者跟agent说人话；引擎那边的代码是coding agent写的。你要做的是知道这件活该交给哪一棒，然后把这句话说清楚。

这个判断本身没法外包。它是这一节全部的内容。

§16 从选题到资产清单

From Topic to Asset List

这一节结束的时候，你手上会有一张表：这个项目需要几件模型、分成哪几类、哪些交给Tripo生成、哪些交给代码长出来。这张表是整个项目里唯一能在动手之前就写完的东西，也是后面每一步的依据。



一张写完了的资产清单长什么样：P2.0那一批40件的逐件渲图，名字和三角面数直接烧在图上（桥28到32k，建筑24到29k，人物15到16k，街具10到13k）。这是第三批，不是开头那23件。我把它放在这一节最前面，是因为清单最后就该长成这个样子：一眼能点数、一眼能对账、一眼知道谁胖谁瘦。 P2.0重生成/P2.0资产总览.png ，2026-09-12。

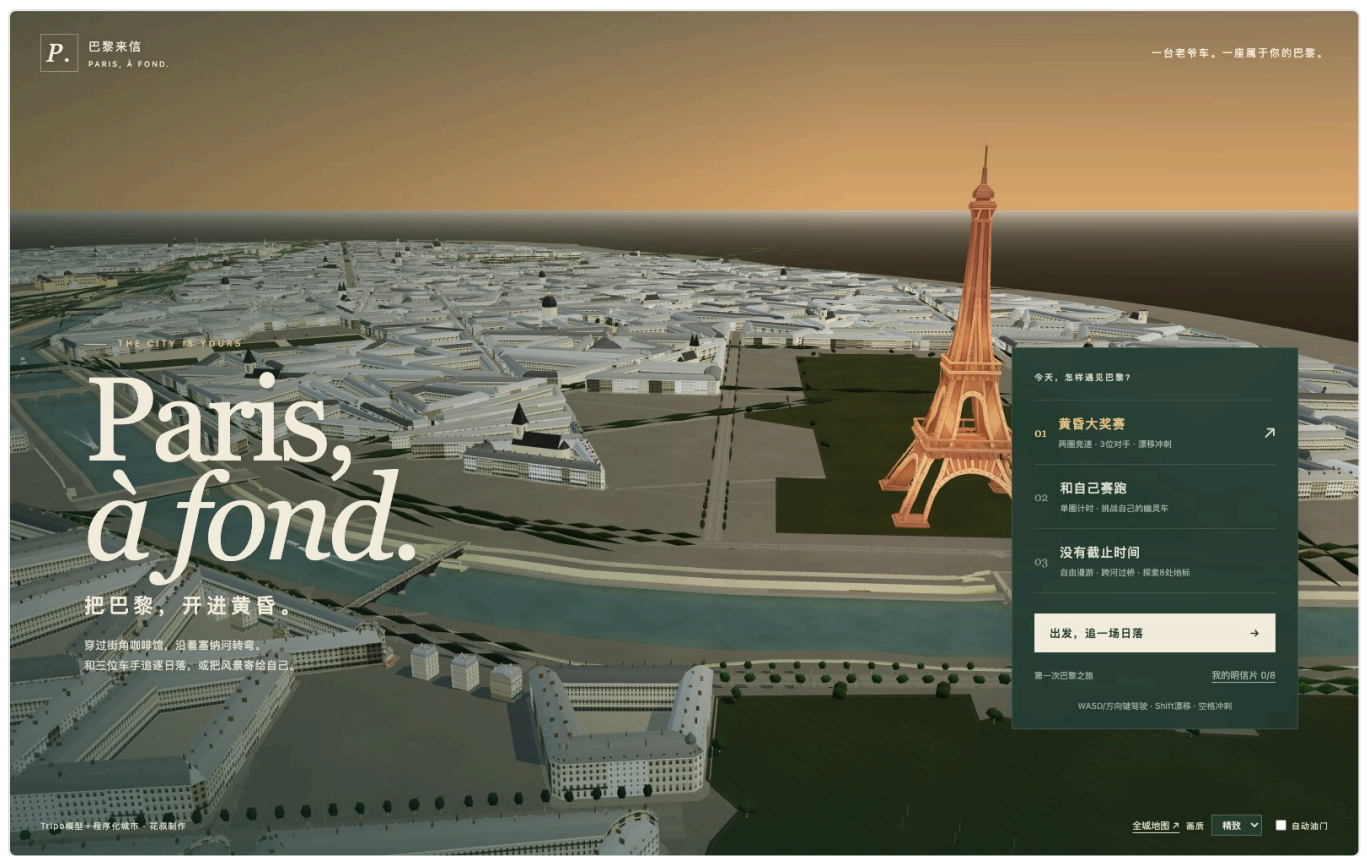
先把话说在前面：这座城里只有23件是生成的

我最后做出来的东西叫「巴黎来信」：一座能开车的巴黎，有塞纳河、有24座桥、有竞速也有自由漫游，手机上能玩，打包成静态网站能发。

但这座城不是Tripo生成的。

Tripo生成的只有23件模型：铁塔、凯旋门、圣心堂、三台车、一批街边建筑、五个行人、几件街具。剩下的那些：1375个街廓、24座桥、6975米河道、河堤、双岛、游船和它拖出来的尾迹、水面反射，全是程序化代码生

成的，来自我自己2026年8月做的另一个项目「巴黎3D地图」，地理数据源自OpenStreetMap，做过艺术化改编。卢浮宫、圣母院、圣礼拜堂也是代码画的，不是模型。



这一张把分工讲得最清楚：中间那座暖褐铜色的铁塔是Tripo生成的GLB，它背后那一整片白色的城市、河道和桥，全是程序化代码画出来的。「巴黎来信」首页，2026-09-14复核截图。

我把这条说在最前面，是因为它太容易被说糊。做完一个这样的demo，最省事的说法是「我用AI生成了一座巴黎」——那句话是假的，而且一旦说出口，读者照着做一定会撞墙。这条分工线本身就是这一章最值钱的内容，它决定了你要花多少积分、等多久、以及最后跑不跑得动。

口径：不能把整座城市称为Tripo生成，也不能把静态行人的摆动说成骨骼绑定。哪一件是生成的、哪一件是代码画的，项目里有逐件的清单和来源哈希（city-reference/source-manifest.json）。

还有一件事必须在这儿一次性说清，不然你往后翻会撞见三个不一样的数：23、25、29，数的不是同一批东西。

数	数的是什么	哪里用它
23	走资产加载器、逐件装进城里的Tripo模型，也就是本节这张清单	本节、§ 18的标题、§ 19的验收行「23件本地资产」

数	数的是什么	哪里用它
25	玩家在城里真能看见的Tripo网格：23件，加第二批上车的两款树。 树是注入城市层的树实例池的，不走资产加载器，所以不在这23件里 （这两款当时是梧桐 tree-01/02-plane，09-12被P2.0的七叶树整体换掉，原因见 § 18）	§ 17批量清洗那一轮跑的就是这25件
29	前两批实际生成并落盘的独立网格总数。硬盘上是31个GLB，其中 pedestrian-01 有静态、绑骨、动画三个文件，是同一个网格，去重得29	§ 04数面数那一轮（16711到46196面，中位18509）

29比23多出来的6件，是第二批里生成并下载成功的那几件：三棵树、三件街具。其中只有两款梧桐最后接进了游戏（那就是25的来历），剩下四件躺在目录里没找到槽位。第二批清单上原本列了更多，但列在清单上和落到硬盘上是两回事，这一节后面会讲。

后来还有第三批，一口气40件，也就是本节开头那张总览图，后面单开了一段讲；它不在上面那三个数里的任何一个，因为29那一轮数面数的时候它还没生成。

这三个数各有各的量法，不能混用，也不用去凑一个「对的」数——它们回答的是三个不同的问题：装了几件、看得见几件、一共生成了几件。你自己做项目的时候也会同时有这三个数，提前知道它们会对不上，比事后去对账省事。

为什么是巴黎

选题这件事，我的判据只有三条，都很土。

第一条：观众不需要你解释它是什么。奥斯曼公寓的弧形转角、锌板孟莎屋顶、黑色锻铁阳台，加一座铁塔和一台雪铁龙2CV，画面里出现半秒钟就没人会问「这是哪」。做3D demo最怕的不是模型糙，是观众看了五秒还在猜你做的是什么地方。辨识度高的选题，等于你把讲解的预算省下来，全花在手感上。

第二条：风格能统一。巴黎的好处是它天然有一套配色：奶油色石材、藏青灰蓝屋顶、黑铁、黄铜。一批资产要批量生成，最怕的是每件单看都不错、摆到一起互相打架。一个有明确时代和色系的题材，等于提前帮你把提示词里的风格段落写死了（这一段怎么写，§ 09有完整模板）。

第三条：我手上已经有一半。前面提过的那个程序化巴黎城市层是我自己做的，能直接复用。选题的时候优先选你已经有一半材料的那个。说是偷懒也行，但省下来的生成预算全压在真正缺的那一半上了。



这一帧里没有任何Tripo模型：屋顶、烟囱、街道纵深、远处的穹顶全部来自程序化城市层。这是「我已经有的那一半」长什么样。QA巡查截图，2026-09-10。

还有一条是我从市场数据里得到的，得单独说。做这个项目之前我扫过X上101条问题帖子，点赞过千的有87条。有两条结论直接影响了选题的形态。一是爆款的分界线在于观众能不能点进去自己玩：可交互作品更容易冲进最高那一档，静态渲染的高赞帖大量堆在1000到5000赞之间（这不等于静态一定输，纯Blender的别墅场景就拿过15061赞）。二是审美疲劳已经从高赞账号开始了，Greg Isenberg (@gregisenberg) 那条3331赞的帖子原话是「我见过100多个人发GPT-6 Astra做3D游戏的demo，挺酷的，但老实说那不是我一直在想的部分」(<https://x.com/gregisenberg/status/2096311243652952229>)。

这两条合起来的意思是：再做一个「我也做了个3D demo」已经不新鲜，差异化只能落在分工、边界和真实成本上。所以这一章不只讲我做了什么，更多篇幅花在讲每一步花了多少积分、哪一步靠代码、哪一步AI做不到。

我动手前唯一的参照，是创作者海辛Hyacinth (@ring_hyacinth) 的一个公开案例，原帖在这里：

https://x.com/ring_hyacinth/status/2096704996381266424

我当时给自己定的目标是做得比它更完整。但这句话到今天也只是目标，我没有统一量表，也没做过对照的用户测试，所以「提升1000%」「超过100%」这种话不能当实测结论。项目文档里我把这条单独写了一行，防止自己后来忘了。

那条线画在哪：会开到跟前的才生成

这是全书我最想让你记住的一条判据，也是我在片子里唯一拿出来让观众抄走的一条：

你开车会开到跟前细看的東西，用Tripo做；你只会远远瞥一眼的，代码画就够了。这样既保住了近景的细节，也不会让帧率掉下来。

写清单的时候，这句话就是你的分栏依据。再补一条管正确性的：尺寸必须精确、后面要参与计算的件，哪怕它就在车轮底下，也归代码。

推荐

交给Tripo生成

车会开到跟前、观众会盯着看的东西；有复杂形体和材质、写代码写不出来的东西。

→ 铁塔的格构、奥斯曼立面的阳台栏杆、2CV的车头、行人的衣褶

不推荐

交给代码长出来

要铺满整座城、靠规则就能生成的东西；尺寸和位置必须精确、后面还要参与碰撞计算的东西。

→ 1375个街廓、24座桥的桥面高程、6975米河道、路网

这条线在游乐园那一轮又原样走了一遍，而且因为游乐设施天然分得开，分得比巴黎还干净：**木马、摩天轮的吊舱、飞椅、茶杯、鬼屋是Tripo生成的；转的圈、轨道、转轴这些结构件是代码写的。**游客会趴到木马跟前看鬃毛和缰绳，但没人会去数摩天轮那圈钢架有几个接头；反过来，那个圈只要不是正圆，所有人一眼就看出来不对（那一轮的完整分工表在 § 14）。同一条判据，换个题材照样用。

为什么不能反过来？算一笔账就清楚了。那一批资产我实测下来是一件40积分（当时的账本记得很清楚：2190积分生成一棵树之后剩2150。40是4K贴图那一档，默认的8K档要50，整条阶梯在 § 07）。1375个街廓如果每个都生成，就是55000积分，专业版十八个月的额度。**而且得排队等**：2026-09-10做巴黎那批资产的时候（高精度模型、4K贴图那一档），我盯着其中一个生成任务过了20多分钟还没返回，当天的开发笔记里原样记着这一行——「生成很慢：一个任务实测20+分钟仍未完成（倒计时显示不准）」（[paris-demo/迭代笔记-细节扩展-20260910.md](#)）。这是**单次观察，不是耗时基准**，我没有第二笔复现，正经的耗时口径在 § 05和 § 07。界面上那个倒计时是估算，别把它当承诺：提交完就走开，Pro的10个并发是拿来一次丢一批、回头统一收的（§ 07）。不过真正让「每个街廓都生成」这件事不成立的，是上面那个数量级，等多久只是顺带。

更要命的是，生成出来的街廓没法用。街廓要参与碰撞计算、要和路网对齐、要按桥面实际高程放车，这些都需要精确的数值，而生成模型给你的是一坨好看的网格。**尺寸件用代码、形体件用AI，这条分工不是成本妥协，是能力边界。**



开到铁塔底下的时候，格构桁架的密度是代码写不出来的，这就是「形体件」该交给生成的理由。QA巡查截图，2026-09-10。

23件是怎么定下来的

我后来反复复用的模板是五分类：**建筑池/地标/街道小品/人物/载具**。巴黎这一批因为地标只有三件，我把建筑池和地标并成了一栏，落到纸上四类。这套分法基本能套所有「AI生成城市场景」的需求，换个城市也一样用。

类别	件数	具体是什么	为什么是这几件
建筑与地标	12	转角公寓、高阁楼公寓、面包店、花店、报刊亭、地铁口、旧书摊、咖啡馆门面、奥斯曼建筑、凯旋门、圣心堂、铁塔	前9件是可以循环铺街的「建筑池」，后3件是唯一摆放的地标，摆放规则完全不同
车辆	3	雪铁龙2CV、老式出租车、载货三轮	一台给玩家开，两台当街景，多一台都是浪费
行人	5	西装男、连衣裙女、骑行者、遛狗人、街头画家	五种够让一条街不重样；后来只有其中一种被绑了骨骼，说明见 § 18
道具	3	咖啡馆桌椅、长椅、花坛	穿插用的密度件，数量少但出现频率最高

建筑池之所以要留9种，是因为它们要沿街循环复用。真正摆上赛道的是其中7种，代码里就是一行 `buildingKeys[i % 7]`，轮着来，每种出现两次。开着车其实能认出来，这是当时一个明确的短板。

核心建议

定建筑池的数量，不看「够不够好看」，看「玩家在一次完整游玩里会经过它几次」。经过两次就能认出来，说明池子太小。

四类各挑一张输入设计图放在这里，你能直观看到风格是怎么统一的：同一段风格描述、同一个浅灰背景、同一个四分之三视角，只换「画面内容」那一段。



建筑类输入图：`building-01-corner.png`，弧形立面转角、铸铁阳台、锌板孟莎屋顶。由Codex按 `jobs.jsonl` 里的提示词批量生成，2026-09-09。



载具类输入图：car-01-taxi.png。同一套风格描述，只换了画面内容那一段。



人物类输入图：pedestrian-05-painter.png。低多边形的面片感是提示词里要的，不是失败，它让后面减面的时候有余地。



道具类输入图：`prop-02-bench.png`。道具件小但出现频率最高，反而最值得单独生成。

清单要先于生成写完

我的做法是：在跑任何一次生成之前，先把整张清单写成一个 `jobs.jsonl` 文件，一行一件。一个资产=一行 JSON =一张设计图=一个GLB =一个文件名，全程不换名字。

文件名长这样：

```
building-01-corner    building-02-tall-attic    building-03-boulangerie
car-01-taxi    car-02-cargo-tricycle    citroen-2cv
pedestrian-01-man-suit    .....    pedestrian-05-painter
prop-01-cafe-table    prop-02-bench    prop-03-planter
```

这个前缀不是为了好看。后面批量清洗的脚本要按资产类型分配减面率，判断依据就是文件名前缀（`pedestrian` 和 `car` 敢砍，`building` 和 `tree` 保守）。命名规范在这里就是给脚本用的接口。§ 17会讲这个判断在什么情况下会判错。



五个行人里的一个，在游戏里的实际观感。她身后那个绿色的报刊亭也是Tripo生成的，脚下的石板路和远处的树是程序化的。QA巡查截图，2026-09-10。

清单不是一次写完的

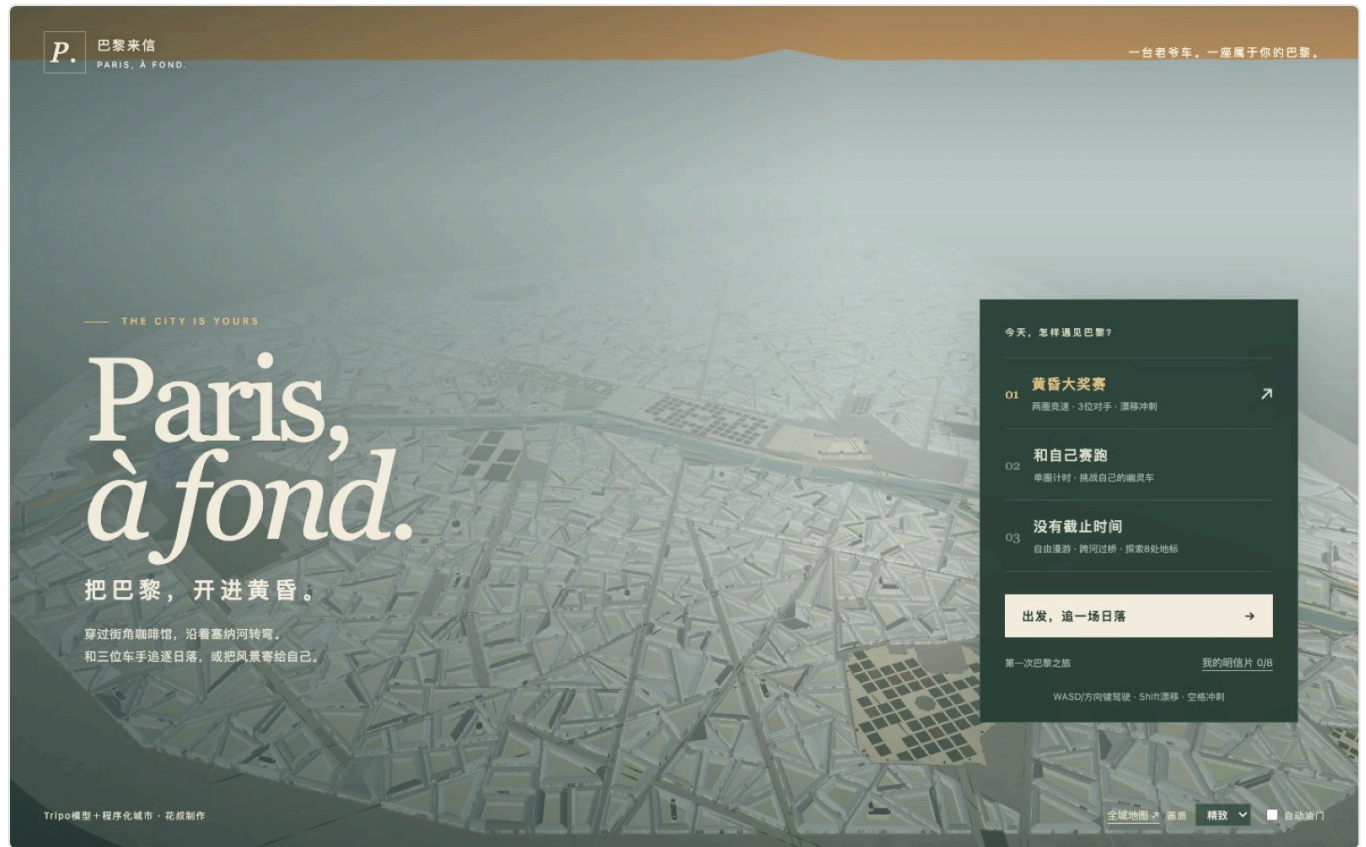
第一批23件跑完、游戏能开起来之后，我做了一轮巡查，96张截图逐帧看。最刺眼的短板不是建筑不够精细，是树。全城1.6到2.3万株行道树全是程序化的深绿色方块，开在街上抬头就是一排绿色积木。

于是有了第二批清单，判据只有一条：哪个东西在画面里出现得最多、同时做得最差。

类别	新增	理由
树 ×3	梧桐（绿）、梧桐（秋黄）、修剪球形	最大短板，全城1.6–2.3万株都是程序化方块
街具 ×4	华莱士喷泉、莫里斯柱、鸽子群、冰淇淋车	巴黎标志物，程序化版本太简陋
人物 ×3	手风琴艺人、气球小孩、情侣	五种行人不够，一条街上重样
载具 ×2	送货面包车、踏板摩托	三台车撑不起街景密度

这张表列了12件，真正生成并下载下来的只有6件：三棵树，和华莱士喷泉、莫里斯柱、鸽子群这三件街具。剩下的卡在下载环节没落盘（长链接和批量下载的坑在 § 10）。这6件最后接进游戏的又只有两款梧桐，另外四件没找到能放的槽位。

所以这一批对三个数的贡献是这样的：给硬盘上添了6个独立网格（23变29），给城里添了2件（23变25），给清单上添了12行。但那两棵树带来的画面变化，比前面九栋建筑加起来都大。



首页背景是整座城的俯瞰，右下角能看到画质切换和「全城地图」入口。这张图里绝大部分面积是程序化的，Tripo资产只在贴近地面时才看得见。2026-09-10。

注意

别把清单当成一次性的产物。第一版清单只需要撑到「游戏能跑起来」，真正该生成什么，要等你在游戏里开一圈之后才知道。我第二批的判据完全来自那96张巡查截图，不是坐在桌前想出来的。

第三批：一次补40件，能进城的只有8个槽位

第二批那几棵树把画面顶上去之后，清单又长了一轮。这一轮我换成智能网格（P2.0）批量重做，一口气出了**40件**，就是这一节开头那张总览图。分类和前两批一个路子，但缺口填得更准：

类别	件数	具体是什么	补的是哪个缺口
桥	3	亚历山大三世桥、艺术桥、新桥	全城24座桥原来全是程序化的，而桥恰恰是你一定会开上去、还会停在上面拍照的地方
地标	6	歌剧院、卢浮金字塔、先贤祠、奥赛钟楼、红磨坊、圣母院西立面	原来只有铁塔、凯旋门、圣心堂三件，地标密度撑不起一座城
船	2	塞纳河游船、驳船	河面上原来只有程序化的船影和尾迹

类别	件数	具体是什么	补的是哪个缺口
载具	6	RATP公交、警车、送货车、Vélib公共单车、DS轿车、踏板摩托	三台车撑不起街景密度，公交和警车还能给街道加叙事
行人	10	咖啡馆侍应、警察、拎法棍的上班族、戴贝雷帽遛狗的老人、慢跑者、骑滑板车的小孩、举相机的游客、手风琴艺人、气球小孩、情侣	7个各不相同的人物件加上前面三件，这是为了终结「满街都是同一个人」，原因见 § 18
建筑与街具	13	奥斯曼转角咖啡馆、阳台公寓、新艺术公寓、旧书摊墙、公交候车亭、Vélib桩、咖啡露台套件、石花槽、铁花槽、七叶树、绿叶与秋黄两款行道树、冰淇淋车	沿街重样感，以及被点名最狠的那棵树

另外还有一个目录叫 glb-1024-新，13件，装的是老槽位的重做版：面包店、花店、报刊亭、凯旋门、圣心堂、铁塔、出租车、载货三轮、西装男、绑骨行人、骑行者、长椅、咖啡馆门面。这一批不是新增内容，是同一个位置换一件更好的。

核心建议

清单分两个目录放这件事值得抄：「新增的」和「重做的」不能混在一起。新增的要找槽位，重做的已经有槽位、只要对齐朝向和尺寸。混在一个目录里，你会在替换的时候反复问自己「这件到底是替谁的」。

但这一轮最该说的不是40这个数，是40件里真正装进城里的只有8个槽位（行道树那件是另一条线单独换的，算第9个，过程在 § 18）。而且这几个槽位全是「换掉老件」，不是新开位置，所以前面那个25没变——城里能看见的还是25件，只是其中几件换成了更好的版本。剩下的躺在目录里等下一轮。原因有三类，每一类都是写清单时该预见到的：

一、映射表本身会写错。我在清单里把 building-03-art-nouveau-guimard 标成了「Guimard地铁口」，准备去替换街上那个地铁口槽位。渲出来一看是一栋六层的新艺术公寓楼，整面立面，Tripo那边的项目名也写着是一栋楼。而那个槽位在代码里是按街具摆的、高度3米。**照着清单换，就是把一栋六层公寓压成3米立在人行道上。**清单是人写的，装之前必须拿渲图核一遍。

二、姿态不对，件本身再好也不能上街。警察、慢跑者、举相机的游客这三件，渲出来都是双臂平伸的对称A-pose。原来那五个行人全是自然姿态（手插兜、骑车、画画、遛狗）。**A-pose的人站在人行道上，读起来像橱窗模特。**所以这三件全部排除，包括我原本给街头画家那个槽位提名的游客件。

三、包围盒一样，几何分布不一样。新的咖啡露台套件比旧的桌椅多一根两米高的取暖伞，结果引擎按总高归一化之后，桌椅从一米一掉到半米，旁边站着一米七的行人。这条的量法和修法在 § 17。

三条合起来是同一句话：清单上的一行不等于一件能用的资产。我第一批23件之所以没踩到这些，是因为那批是「先画设计图、再按图生成」，图什么样模型就什么样；到了批量重做这一轮，件是先有的、槽位是后配的，

中间就多出了一道对不上的风险。

这一节的产出

到这里你应该有三样东西：一张分四类的资产清单、一份写好的 `jobs.jsonl`、一条画清楚的分工线（哪些生成、哪些代码）。下一节把这张清单变成能进引擎的文件：逐件生成、批量减面、归一化尺寸，以及我在这条流水线上踩的三个坑。

§17 生成与清洗流水线

The Generate-and-Clean Pipeline

把 § 16 那张清单变成能进引擎的文件，中间有四步：逐件生成、导出GLB、批量减面、归一化尺寸。这一节给你能直接复制的命令，以及我在每一步踩到的坑：一个「减面减不动」的下限，和一个让整排树糊成一片的缩放基准错误。

四步，最贵的那步是等



先给你一个心理预期：**整条流水线里，真正吃时间的是第一步的排队，不是后面的处理。**25个资产（§ 16那23件，加后来补的两棵梧桐）的批量清洗我实测跑完只要131秒，平均5.2秒一个；而生成那一步的耗时是会波动的，我盯过一个任务20多分钟还没返回（高精度模型加4K贴图那一档，出处和「这是单次观察不是耗时基准」那句在 § 16），界面上那个倒计时是估算不是承诺，正经的耗时口径在 § 05。

所以这条流水线正确的跑法是**并行提交、串行清洗**：生成那一步全部丢出去排队，等的时候去写游戏代码，模型陆续回来之后再一把清洗。

第一步：逐件生成

巴黎那23件走的都是网页版的「高精度模型」，一件一件提交。表单长这样：

首页生成入口见 § 01 那张：传一张设计图，左边选高精度模型还是智能网格，右边按钮直接标着这次要花多少积分。

积分账我当时记得很细：生成一棵树之前账户是2190，之后是2150，**一件40积分**。40是4K贴图那一档的价，同一个「高精度模型」默认走8K就是50，几何底价固定15，差额全在贴图档上（2K 30 / 4K 40 / 8K 50，再开Ultra网格65），完整阶梯在 § 07。23件加上中途几次重试，开发笔记当时估的是700到800积分——那是估数不是逐笔加出来的，真按40一件硬算会更高一些。无论按哪个口径，都不到专业版月度额度的三分之一，对一个中等规模的场景生产任务，额度是够用的，不用刻意省。

两个观察值得单独说。

一是成功率高得有点意外。23件用同一套「Codex出设计图 → Tripo图生3D」的流程跑下来，材质质感、比例、可辨识度全部在线，其中留了设计图、能照着复现的19件全部一次通过，没有一件因为明显失败要重跑（23和19这两个数怎么回事，§ 08里说清楚了）。

二是面数落在了一个很舒服的区间。这批资产普遍在1.5到2万三角面，直接就是游戏能用的量级，不需要再走重拓扑那道工序。但我不想把这个说成「面数可控」。准确的说法是默认值恰好落在可用区间，不是我主动调出

来的。同样关掉超清几何精度，镂空精细的奥斯曼建筑天生92.7万面，结构简单的铁塔天生只有1.9万面。面数由对象本身的复杂度决定，不由你写的提示词决定（写「low poly」它也不听）。

资产页（图见 § 03）是并行提交之后的看板：卡片上标着每件已经等了多久，顶部那四个筛选，后面找模型全靠它。

第二步：导出GLB

导出弹层里只有三个选项要管：文件名、格式选GLB、贴图分辨率。

导出弹层见 § 03那张图。格式与贴图分辨率这两个选项在高精度模式下是一样的，贴图分辨率这里选1k，原因见下一段。

贴图分辨率在这一步就要降下来。Tripo默认导8192×8192，17件这样的资产实测吃掉5.0 GiB显存（=十进制的5.38 GB，口径见 § 12），降到1024×1024之后是0.22 GiB。游戏视距下你看不出区别，但显存是实打实的。

导出这一步有两个和模型质量无关、但一定会拦住你的东西。一个是免费档的导出受限：官方对照表里Free那列的 Exports 原文是 15 (H2.5 only)，同列的Private Models和Commercial Use都是❌，所以免费档能试质量、不能交付，全书统一口径见 § 07与附录C。另一个是浏览器的原生「另存为」对话框。如果你像我一样让AI agent代跑，那个系统弹窗是所有浏览器自动化工具的停手线，谁都不该替你点。解法是走API直链取GLB，完全绕开导出按钮，方法在 § 03。

第三步：批量减面

这一步是纯收益，没有理由跳过。我第一遍用的是统一参数：

```
npx @gltf-transform/cli optimize in.glb out.glb \
  --compress quantize --texture-size 512 \
  --simplify true --simplify-ratio 0.6
```

23个资产平均减重65%，37 MB压到13 MB，肉眼看不出画质损失。

`--compress quantize` 这个选择是有理由的：量化不需要额外的运行时解码器，浏览器直接就能播；换成draco或meshopt，都要在Three.js里另挂一个WASM解码器，而在某些沙盒环境里那个解码器本身的加载就会失败。为了一点压缩率换一个可能加载不上的依赖，不划算。

减面的下限，由几何有多光滑决定

统一参数跑完之后我做了一个实验，想知道每类资产到底能砍到多少。方法是对11个资产各跑一遍：

```
npx @gltf-transform/cli simplify in.glb out.glb --ratio 0.1 --error 1
```

ratio从0.5一路试到0.02，记录面数不再下降的那个拐点。那张11行的减面下限表和它的完整读法在 § 04，这里不重抄；两个口径要交代：表里的「原始面数」指进这轮实验时的面数（都已经压过一遍，那棵梧桐的7473刚生成出来是18414面，见下面「树的三个坑」；奥斯曼建筑那行的41106，Tripo直出时是92.7万面）。

拿到流水线上来讲，这张表只有一个生产含义：减面率要按资产类型分配，不能一刀切。人体那类连续曲面砍得最狠，建筑上全是直角窗洞栏杆砍不动，树和铁塔几乎砍不动。判断函数和两条命令都在 § 04。

25个资产跑完的账与 § 04那张表相同：14600 KiB降到9433 KiB（减35%），总耗时131秒、平均5.2秒一个——那5.2秒大部分是 npx 的启动开销，不是计算。

这一轮也抓到一个反例：prop-03-planter 按前缀判成道具、给了0.25的激进比例，结果只减到7652面。它叫 **prop**，几何却像建筑，花坛全是直角结构。文件名前缀是个够用的近似，但它会判错，判错的表现就是「明明给了激进参数却没减下去」。

注意

说清楚一件事：这一轮分级减面是在「已经压过一遍」的那批文件上重跑的实验，结论拿到了，但当前发布包里用的还是第一遍统一参数的那批产物。分级流水线是给下一个项目准备的，不是这个demo的实际构成。这种地方我不想含糊过去。

树的三个坑

梧桐是整批资产里最难处理的一件，三个问题一个个来。



左边原始18414面，右边减面后7473面。ratio给0.1和给0.05结果完全一样，再往下砍就会破坏分枝拓扑，这就是这棵树的底。2026-09-10。

坑一，**optimize** 子命令里的 **--simplify** 对它不生效。必须用独立的 **simplify** 子命令，并且带上 **--error** 参数，减面才真的发生。

坑二，减面有硬下限。18414面用meshoptimizer最多减到7473面，ratio 0.1和0.05的结果一模一样，再减就破坏分枝拓扑。这和上面那张表里95.3%那一行是同一件事。

坑三，三角面预算会爆。城市层原来用程序化实例画树，全城2万株只占9个绘制调用、8.4万三角面。按它原来的近景实例上限直接换成15000面的GLB树，光近景这一档就是225万三角面，是整层预算的26倍。当时的做法是近景只放36株GLB树、中景和远景继续用程序化，这是画面和帧率之间唯一走得通的折中。这个36后来提到110，实测几乎不花钱，见 § 18。

第四步：归一化尺寸

这一步最容易被跳过，也最容易在集成那天炸给你看。

生成出来的GLB，每一件的尺寸基准都不一样：有的模型原点在几何中心，有的在脚底，整体高度也是随机的。引擎那边要的是一个统一契约。我给城市层定的契约是两条：y=0落地、整体高度归一到1个单位，之后由引擎按真实米数缩放。代码就这么几行：

```
const box = new THREE.Box3().setFromObject(g.scene);
const size = box.getSize(new THREE.Vector3());
const center = box.getCenter(new THREE.Vector3());
geo.translate(-center.x, -box.min.y, -center.z); // 水平居中、脚底落到 y=0
geo.scale(1/size.y, 1/size.y, 1/size.y); // 整体高度归一到 1
```

然后我踩了这一节最丢人的一个坑：**缩放基准选错了**。

GLB树的宽高比接近1比1，而原来的程序化树是「宽8高11」的瘦高个。我一开始按整树高度等比放大，冠幅直接放到14米，而巴黎行道树的间距只有8米。结果就是整排树的树冠彼此重叠，糊成一堵绿墙。改成按冠幅缩放（高度系数1.2，冠幅落到8.4米）才对。

核心建议

归一化的时候问自己一句：这个资产在场景里是靠哪个维度和邻居对齐的？行道树靠冠幅对齐间距，建筑靠临街面宽度对齐街廓，人物靠身高对齐门框。**按那个维度归一，不要习惯性地按整体高度。**

同一个坑的第二种长法：包围盒一样，几何不一样

后来批量替换资产的时候，这个坑换了张脸又来了一次，而且这次更阴。

新的咖啡露台套件要去替换街上那套咖啡桌椅。两件包围盒的比例看着都正常，但新件比旧件多了一根两米高的户外取暖伞。引擎按包围盒总高归一化到1.1米，于是桌椅从1.1米掉到约0.5米，旁边就站着1.72米的行人。发现它的办法不是肉眼看，是量**顶点高度的分布**：

件	顶点高度p90落在总高的	意思
旧 prop-01-cafe-table	0.80	九成几何铺满整个高度，总高就是主体高

件	顶点高度p90落在总高的	意思
新 street-03-cafe-terrace-set	0.46	九成几何挤在下半截，上半截是一根细伞杆撑的

判据：p90明显低于0.7，说明这件东西的包围盒高是被一根细杆、一根天线、一顶伞撑起来的，不能拿总高做归一基准。这条一行代码就能算，值得写进你的清洗脚本，让它在装之前自己报警。

这件最后没换。要换的话也只是一行的事：把那个槽位的目标高度从1.1米改到约2.1米，或者干脆把取暖伞裁掉重出一版。但我不想为了凑一件资产去动摆放代码里的常数，那是把资产的问题记到引擎头上。



替换之前：右侧那一排深绿色的几何体就是程序化行道树，全城1.6到2.3万株都长这样。QA巡查截图，2026-09-10。



左右对照：左边是程序化顶点色方块树，右边是接进去之后的Tripo梧桐（带树皮和叶片贴图，绿叶与秋黄两个变体交替）。同一套城市层，只换了近景那一档。2026-09-10。



俯瞰机位下的集成验证。近景这一档换成GLB之后，树冠的层次和叶片形状是程序化几何体做不出来的。2026-09-10。

批量那一档：38件一次过，两道工序

上面那条流水线是给「二十几件」这个量级写的。等到智能网格那批一口气出了38件，有两道工序必须换个做法，不然光是手工点导出就够点一下午。

工序一：降贴图不开Blender，直接改文件

这38件出来已经是GLB了，所以这一道我没走Blender，也没走glTF-transform，而是直接改GLB里的glTF JSON和BIN数据块：只动两样东西，images那几段字节，以及可选的顶点数据。其余字节原样搬过去。这么做的好处不是快（虽然确实快），是「三角面不变、包围盒不变、扩展依赖不变」这三条不需要事后去核验捞回来，构造上它们就动不了。你没碰的字节不会自己变。

	源	产物
件数	38	38
合计体积	534.6 MB	52.4 MB (9.8%)
合计三角面	735,232	735,232（逐件相等）
贴图	每件1张8192 ² JPEG	每件1张1024 ² JPEG, q92、不做色度抽样
doubleSided	全true	全false

	源	产物
扩展依赖	空	空

逐件核验八项，38件全过：三角面不变、顶点数不变、图元全是三角形、带UV和法线、扩展依赖为空、包围盒偏差不超过 $1e-5$ 、单面、贴图确实是 1024^2 。逐件日志一件一个文件存着，出了事能追到具体是哪一件哪一项。

这一批每件只带一张base color贴图，所以降贴图这一步不用管法线图和ORM图的对应关系，一张换一张就完了。要是你手上的资产带多张贴图，记住一条：按原图名一一对应地换，不要「把所有贴图槽位都换成第一张图」。我第一版脚本就是那么写的，出租车只有一张图所以看不出问题，换成带法线图的资产就会把法线图也替成颜色图，而这个错误渲出来还挺好看，不看文件根本发现不了。

顺手修掉的一件事：这38件的 `doubleSided` 全是`true`。它们是用Blender批量导出的，而Blender的FBX导入器默认不开背面剔除，导出到glTF就成了双面，片元开销直接翻倍。我在这一道工序里统一强制成单面，和城里现有的那批对齐。**这个坑不看GLB的JSON发现不了，模型渲出来一模一样。**薄片类资产（招牌、树叶）要留双面，得单独开开关。

降贴图这一步为什么非做不可，看显存账最直观：一张 8192^2 的底图展开268 MB、带mipmap约358 MB，26件就是9.3 GB——这个数不需要讨论，它直接就是「开不起来」；降到 1024^2 之后每件5.59 MB、26件145 MB。完整算法、进位口径、以及为什么显卡工具读数会小约7%，都在§12那笔账里，这里不重算。

一个诚实的交代：1024这一档我是照配方定的，没和512、2048做过A/B，所以我只能说它在这个项目里够用，不能说它是最优解。

工序二：FBX转GLB，让两套读数互相不信任

智能网格带贴图导出给的是FBX，而Web这边吃的是GLB，中间得有一道转换。本机没有现成的命令行工具能读FBX，最后用的是Blender的无头模式：

```
python3 fbx2glb.py taxi-p2-smartmesh-textured.fbx \
  -o out/car-01-taxi.glb --yaw -90 --dump-texture
```

选Blender有一条硬理由：这批FBX的Creator字段写的就是Blender的FBX导出器。用同一个实现读回来，轴向和单位的往返最不容易出岔子。

但转换器自己的读数不能当证据。我让它同时跑三条互不相干的通道去量同一个包围盒：

1 纯Python解FBX

自己解出顶点，手算那个把Z轴朝上转成Y轴朝上的旋转，得到FBX世界包围盒。这条完全不碰Blender。

2 Blender侧读世界包围盒

导入之后直接问Blender尺寸是多少，再按轴向映射回去。

3 解产物GLB

读GLB的顶点数据加节点矩阵，重新算一遍包围盒。

三条通道的最大偏差是**1.0e-08**。这个数的意思是浮点误差级别，也就是三条路量的是同一个东西。出租车那件的实测账：

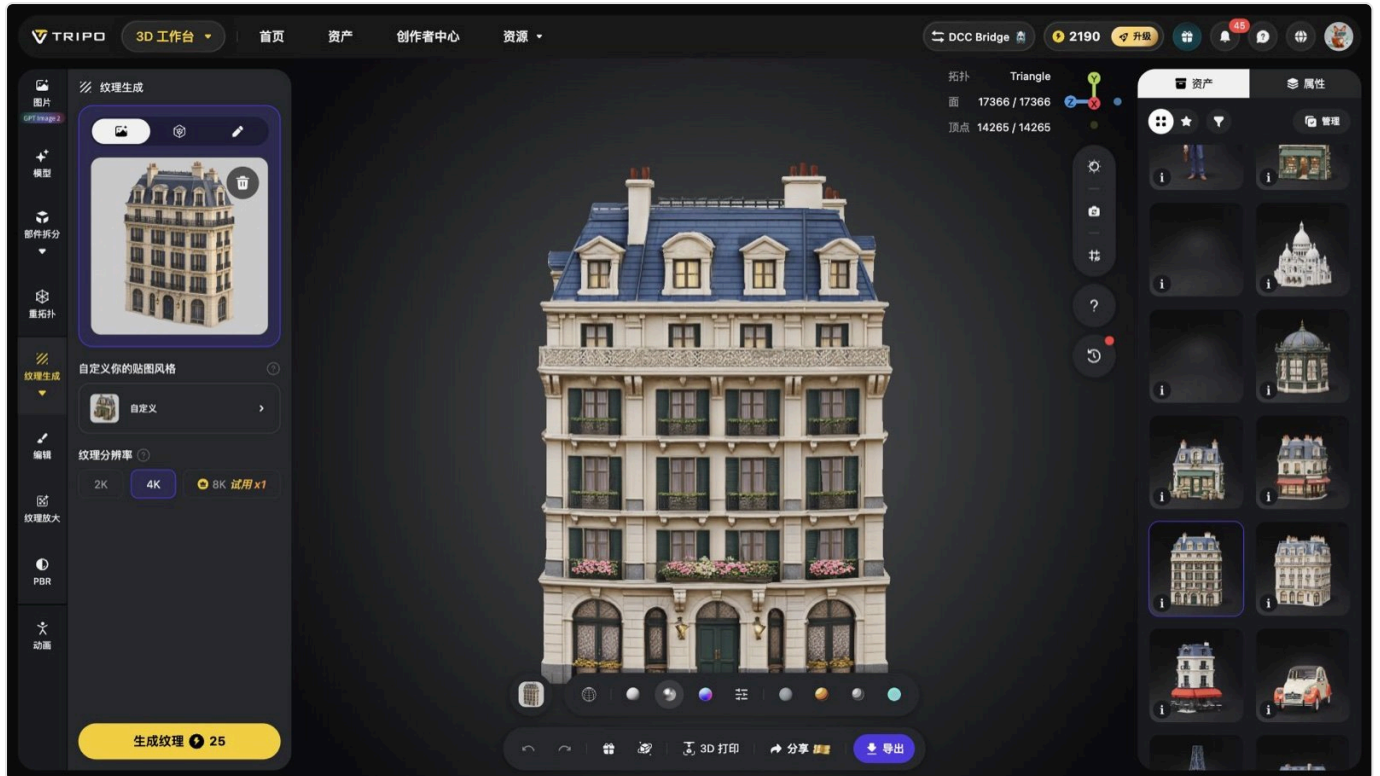
	源FBX	产出GLB
文件体积	7,473,036字节 (7.13 MiB)	747,724字节 (730 KiB)
多边形	5,615 (四边形4,185, 占74.5%)	—
三角面	9,800 (三角化后)	9,800, 逐位相等
贴图	1张8192 ² JPEG, 6.81 MiB	1张1024 ² JPEG, 308 KiB
包围盒	0.516113 × 0.543457 × 0.991699	偏差1.0e-08

注意

包围盒能兜住尺寸，兜不住正反。`--yaw 90` 和 `--yaw -90` 转出来的包围盒完全一样，数字上分不出车头车尾，给反了车就是倒着开。唯一的办法是渲出来看：新旧资产各渲一组视图，并排比，读出差多少度。而且四视图不够，得渲六视图，因为正面和背面在四视图里是分不出来的。这一步没有自动办法，每一件都得自己看一眼。

一个省钱的岔路：同几何换贴图

前面提过，沿街的建筑是七种循环复用，开着车能认出来在重样。解决重样最直觉的办法是多生成几栋楼，但还有一条便宜得多的路：几何不动，只换贴图。



纹理生成面板：左边传一张参考图定义贴图风格，右上角显示几何完全没变（面17366 / 17366，顶点14265 / 14265），百叶窗变了深绿、窗台长出花箱。这次走的是4K档：加自定义参考图25积分，不带参考图20积分（默认的8K档按钮上写的是30，界面见 § 05）。2026-09-10。

实测下来几何是零改动的，17366面进、17366面出。对游戏来说这是双重的好处：消除重样感，同时几何还能在多个实例之间共享，省显存。给核心的几栋建筑各配一两张贴图变体，比再生成几栋楼划算得多。

这一节的产出

一个 `assets/` 目录，里面是命名规范、尺寸归一、面数在预算内的GLB。下一节把它们接进引擎：五个文件各管一摊，加上车辆手感、会走路的行人，和两个把我卡了半天的坑。

资产	ratio	面数	→	面数	体积 KB	→	KB
building-01-corner	0.45	17043	→	7650	618	→	439
building-02-tall-attic	0.45	17177	→	7717	602	→	433
building-03-boulangerie	0.45	16696	→	7504	567	→	388
building-04-fleuriste	0.45	17232	→	7742	575	→	393
building-05-kiosque	0.45	18101	→	8108	534	→	358
building-06-arc	0.45	16619	→	7452	514	→	346
building-07-sacre-coeur	0.45	18734	→	8368	529	→	347
building-08-metro	0.45	17849	→	7989	566	→	377
building-09-bookstall	0.45	16961	→	7609	547	→	375
car-01-taxi	0.25	18083	→	4451	487	→	245
car-02-cargo-tricycle	0.25	18493	→	4528	561	→	292
citroen-2cv	0.25	18078	→	4503	525	→	284
eiffel-tower	0.75	19424	→	14553	700	→	600
haussmann-building-lowpoly	0.45	41106	→	18446	1884	→	1336
pedestrian-01-man-suit	0.25	19042	→	4751	443	→	204
pedestrian-01-walk	0.25	19007	→	4744	584	→	285
pedestrian-02-woman-dress	0.25	18602	→	4636	468	→	233
pedestrian-03-cyclist	0.25	19105	→	4749	583	→	314
pedestrian-04-dog-walker	0.25	18530	→	4584	449	→	211
pedestrian-05-painter	0.25	18542	→	4614	465	→	220
prop-01-cafe-table	0.25	18746	→	4670	507	→	260
prop-02-bench	0.25	16320	→	4038	451	→	231
prop-03-planter	0.25	17049	→	7652	595	→	389
prop-cafe-shop	0.25	17653	→	5640	583	→	346
tree-01-plane-green	0.75	7473	→	7123	390	→	380
tree-02-plane-autumn	0.75	7913	→	7143	457	→	432
合计：26 个资产 15184KB → 9718KB (减 36%) 总耗时 199s 平均 7.6s/个 alchain@MacBook-Pro-127 三模式实验 %							

E2批量清洗脚本跑完的真实终端输出，逐件一行、末尾一行合计（截图裁掉了窗口两侧的空白，字放大了一倍多，26行一行没少）。这张是2026-09-11重跑的，所以是26个资产不是上面表里那25个——素材目录比当初多了一个绑骨版行人 pedestrian-01-walk.glb。这一轮的合计是15184 KiB → 9718 KiB (减36%)、总耗时199秒、平均7.6秒一个；比当初那轮慢，是因为多了一件、而且这次机器上还跑着九个demo服务。上面表里那组25 / 131秒是当时的原始记录，我没有用今天这轮去覆盖它。

assets				optimized			
名称	修改日期	大小	种类	名称	修改日期	大小	种类
building-01-corner.glb	前天 09:00	1.9 MB	GL 传输...二进制	building-01-corner.glb	前天 13:44	633 KB	GL 传输...二进制
building-02-tall-attic.glb	前天 09:09	1.9 MB	GL 传输...二进制	building-02-tall-attic.glb	前天 13:44	617 KB	GL 传输...二进制
building-03-boulangerie.glb	前天 09:16	1.6 MB	GL 传输...二进制	building-03-boulangerie.glb	前天 13:44	581 KB	GL 传输...二进制
building-04-fleuriste.glb	前天 09:24	1.6 MB	GL 传输...二进制	building-04-fleuriste.glb	前天 13:44	589 KB	GL 传输...二进制
building-05-kiosque.glb	前天 09:34	1.5 MB	GL 传输...二进制	building-05-kiosque.glb	前天 13:44	547 KB	GL 传输...二进制
building-06-arc.glb	前天 09:55	1.5 MB	GL 传输...二进制	building-06-arc.glb	前天 13:44	527 KB	GL 传输...二进制
building-07-sacre-coeur.glb	前天 10:09	1.5 MB	GL 传输...二进制	building-07-sacre-coeur.glb	前天 13:44	542 KB	GL 传输...二进制
building-08-metro.glb	前天 10:17	1.6 MB	GL 传输...二进制	building-08-metro.glb	前天 13:44	581 KB	GL 传输...二进制
building-09-bookstall.glb	前天 10:26	1.5 MB	GL 传输...二进制	building-09-bookstall.glb	前天 13:45	561 KB	GL 传输...二进制
car-01-taxi.glb	前天 11:55	1.2 MB	GL 传输...二进制	car-01-taxi.glb	前天 13:45	499 KB	GL 传输...二进制
car-02-cargo-tricycle.glb	前天 12:03	1.5 MB	GL 传输...二进制	car-02-cargo-tricycle.glb	前天 13:45	575 KB	GL 传输...二进制
citroen-2cv.glb	前天 00:29	3 MB	GL 传输...二进制	citroen-2cv.glb	前天 13:45	538 KB	GL 传输...二进制
eiffel-tower.glb	前天 00:37	1.7 MB	GL 传输...二进制	eiffel-tower.glb	前天 13:45	717 KB	GL 传输...二进制
haussmann-building-lowpoly.glb	前天 00:35	4.7 MB	GL 传输...二进制	haussmann-building-lowpoly.glb	前天 13:45	1.9 MB	GL 传输...二进制
optimized	前天 13:45	--	文件夹	pedestrian-01-man-suit.glb	前天 13:45	455 KB	GL 传输...二进制
pedestrian-01-man-suit.glb	前天 10:38	1.1 MB	GL 传输...二进制	pedestrian-02-woman-dress.glb	前天 13:45	480 KB	GL 传输...二进制
pedestrian-02-woman-dress.glb	前天 10:48	1.3 MB	GL 传输...二进制	pedestrian-03-cyclist.glb	前天 13:45	597 KB	GL 传输...二进制
pedestrian-03-cyclist.glb	前天 11:09	1.6 MB	GL 传输...二进制	pedestrian-04-dog-walker.glb	前天 13:45	461 KB	GL 传输...二进制
pedestrian-04-dog-walker.glb	前天 11:21	1.2 MB	GL 传输...二进制	pedestrian-05-painter.glb	前天 13:45	477 KB	GL 传输...二进制
pedestrian-05-painter.glb	前天 11:43	1.2 MB	GL 传输...二进制	prop-01-cafe-table.glb	前天 13:45	519 KB	GL 传输...二进制
prop-01-cafe-table.glb	前天 13:23	1.4 MB	GL 传输...二进制	prop-02-bench.glb	前天 13:45	463 KB	GL 传输...二进制
prop-02-bench.glb	前天 13:30	1.2 MB	GL 传输...二进制	prop-03-planter.glb	前天 13:45	609 KB	GL 传输...二进制
prop-03-planter.glb	前天 13:43	1.5 MB	GL 传输...二进制	prop-cafe-shop.glb	前天 13:45	597 KB	GL 传输...二进制
prop-cafe-shop.glb	前天 09:00	1.6 MB	GL 传输...二进制				

清洗前后的两个目录并排：左边 assets/，右边 assets/optimized/，都开着「大小」列。同名资产一眼可比，最大的那件从 1.9 MB降到633 KB。

§18 组装：把23件模型变成一个能开的城

Assembly: From 23 Files to a Drivable City

这一节讲怎么把清洗好的GLB接进引擎，五个文件各管一摊；也讲我作为一个不写代码的人，是怎么指挥编程agent干这件事的，包括我说过的原话、以及「不许做」那一栏为什么比「要做」那一栏更值钱。

先看它长成了什么样



自由漫游模式。左下角是街区小地图，右下角是速度和冲刺储备，中间那个绿色浮标是下一个地标的方向与距离。2026-09-10。

这个东西有三种玩法（两圈竞速、单圈幽灵计时、自由漫游），一条约2.12公里的赛道，8处可探索地标，能拍照导出明信片，手机触屏能玩。全部跑在浏览器里，不依赖任何CDN。

五个文件，各不认识对方

整个项目的代码分成五个模块，划分依据是各自不需要知道什么：

文件	管什么	它不需要知道
engine.mjs	固定步长物理、赛事流程、漂移奖励、进度计算	这座城是巴黎，还是任何别的城
city-map.mjs	赛道路线、街廓碰撞、河岸保护、桥面高度	车是怎么开的

文件	管什么	它不需要知道
world.mjs	Tripo资产的加载与摆放、渲染、行人动画	比赛规则
city-loader.mjs	把旧的程序化城市模块接进来	游戏本身
app.mjs	输入、界面、存档、拍照、地图	物理怎么算

这个划分是被逼出来的。项目有一个特殊约束：**城市层从另一个项目原样搬过来，那个项目必须只读**。所有适配只能发生在边界上：坐标系从「北向+Z」转成「北向-Z」、r160的立面图集UV、水面shader的输出格式、六层构建回执的强制核对，全部塞进 city-loader.mjs 一个文件里。

核心建议

接入一个不属于你的代码库时，把所有适配逻辑收在一个文件里。这样当上游更新、或者你需要证明「我没改过它」的时候，只有一个地方要看。我把复制来源和每个文件的SHA256哈希都记在了 city-reference/source-manifest.json 里。



这一帧同时出现三层：脚下的石板路和两侧的立面来自程序化城市层，右上角的铁塔是Tripo的GLB，画面比例和雾色是渲染层统一调的。QA巡查截图，2026-09-10。

我是怎么指挥的

我不写代码，这个项目的每一行都是AI写的。所以怎么指挥这件事，对我来说比怎么实现重要得多。

我的做法是每一轮开工之前先写一张纸，就五栏：成品长什么样（我的原话逐字抄进去）、背景、**不许做**、三条可判真假的过关标准、什么时候该问我。

巴黎这一轮，第一栏是这么写的：

「我们现在这个游戏还是不够好，请作为出色的游戏策略，以及产品经理、架构师角色，再去疯狂优化，提升1000%+的产品体验。我们做出来的东西需要比brief中提到的海辛的案例出色100%以上」

地图那一轮是这么写的：

「倒也不是不能用，但是地图的规模，以及整体细节差很多。我觉得现在优质的模型部分和游戏机制可以保留并进一步提升。但是地图的实现方式，更多的桥梁和水面的细节，建筑的布局，你可以参考一部分这个项目的实现」

加资产那一轮是这么写的：

「再去疯狂测试和迭代吧！！！！我们的游戏需要优化，也需要更多可用的，让游戏细节更丰富的模型」

行人那一轮是这么写的：

「肯定要做骨骼能动的游戏，而不是一整个模型在平移，那毫无意义」

你看这四句的共同点：**全是方向和判据，没有一句是实现**。我没说用什么算法做漂移，没说行人的动画该怎么驱动，没说城市层该怎么接。我说的是哪里不够好、保留什么、这个方向能不能参考、什么样算毫无意义。

这不是因为我懂得少所以只能这么说。而是我说得越具体，agent的方案空间就越小，而它在实现层面的判断力其实比我强。我该占住的是另一头：这东西给谁玩、玩到哪一步算好、哪些事绝对不许做。

「不许做」那一栏

五栏里我花时间最多的是第三栏。巴黎这轮写了五条：

- 不把「1000%」和「100%」写成验证结果
- 不改其他项目（城市层那个参考项目必须只读）
- 不付费生成、不擅自发布
- 保留旧版
- 静态GLB不冒充骨骼动画

这五条里，第一条和第五条**直接决定了这本书里哪些话能说**。如果当时没写死，今天我大概率会拿着「提升1000%」当成果去讲，而那是一句没有量表、没有对照测试支撑的话。第五条更要命：如果允许把位移加上起伏说成「走路动画」，我就不会去测真正的骨骼绑定，而那恰恰是这个项目后来最有价值的一段。

注意

给agent写任务的时候，「不许做」比「要做」更值钱。「要做」它能自己推，「不许做」它推不出来，因为那一栏里装的是你的责任边界，不是技术判断。

车辆手感：一个变量拆成两个

第一版车开起来很怪。不是慢、不是飘，是「太跟手了」。方向一打，车就立刻整个换方向，像用鼠标拖一个图标，没有任何重量感。

原因是当时只有一个 `heading` 变量，同时负责「车头指向哪」和「实际往哪走」。转向输入100%立刻体现在位移上，没有任何滞后。

改法是把它拆成两个：

```
// heading: 车身朝向，转向输入直接改它
p.heading += p.steer * (drifting ? 2.3 : 1.65) * ... * dt;

// moveDir: 实际移动方向，每帧用一个"抓地力"系数向 heading 追赶
p.moveDir += angleDelta(p.moveDir, p.heading) * (1 - Math.exp(-(drifting ? 3.0 : 11) * dt));

p.x += Math.sin(p.moveDir) * p.speed * dt;
p.z += Math.cos(p.moveDir) * p.speed * dt;
```

关键在那个抓地力系数：**正常驾驶是11，按住漂移键掉到3.0**。数值高的时候 `moveDir` 追得紧，车贴地；数值低的时候 `moveDir` 明显落后于 `heading`，车身指着一个方向、实际往另一个方向滑，这就是甩尾。

一个改动同时解决了太跟手和不会漂移两个问题，是这个项目里性价比最高的一条经验。漂移超过0.35秒松开给冲刺奖励，超过1.05秒给大奖励，连击还会翻倍加分，这套东西全部长在那两个变量的差值上。



漂移状态：车身朝向和移动方向分开之后，地面上才会留下这条拖痕，右下角提示「完美！松开漂移冲刺」。83 km/h，QA巡查截图，2026-09-10。

同一个思路还救了另外两处。碰撞不需要物理引擎：每个静止物体注册一个圆心加半径，每帧检查距离、重叠就推出去，几百个碰撞体对现代浏览器毫无压力。AI对手不需要导航网格：赛道是固定闭环，路点巡航就够了，朝向做平滑插值，转弯处天然带弧度。通用寻路是为「任意起点到任意终点」设计的，用在这里是高射炮打蚊子。

后来又补了一刀：高速转向要收敛

手感这条线后面还改过一次。原来的转向角速度只有一个「停着不能原地打转」的限制，7 m/s就到顶了，也就是25 km/h和83 km/h打方向的角速度完全一样。83 km/h下转弯半径只有13.9米，车发飘、点一下方向就画圆，高速段没有任何重量。更要命的是漂移比抓地只快了不到四成，漂移在手感上没有存在的必要。

改法是给转向乘一个随速度衰减的抓地系数，13 m/s以下完全不动（低速灵活是这台车的性格，不能碰），高速段再加一条「满舵会蹭掉一点速度」。实测的稳态转弯半径：

速度	改前	改后	改后 · 漂移
7 m/s (25 km/h)	4.2 m	4.2 m	4.2 m
12 m/s	7.3 m	7.3 m	5.2 m
16 m/s	9.7 m	11.0 m	7.9 m
20 m/s	12.1 m	15.7 m	11.2 m

速度	改前	改后	改后 · 漂移
23 m/s（顶速83 km/h）	13.9 m	19.5 m	14.0 m
34 m/s（冲刺122 km/h）	20.6 m	28.8 m	20.7 m

低速一模一样，顶速收到71%。**漂移的价值第一次成立**：23 m/s下漂移14.0米对抓地19.5米，差28%，「高速弯要么减速要么漂」这句话从手感上真的兑现了。

这一改差点被验错，而且错得很彻底。那个故事放在 § 19，因为它是关于验收方法的，不是关于车的。



三台AI对手车用的是路点巡航，不是导航网格。它们在这条固定闭环上能完整跑完两圈，30 / 60 / 120 fps下的完赛时刻逐项相等。QA 巡查截图，2026-09-10。

行人：从滑行到真的在走

第一版的行人是会动的静态雕塑。Tripo那批没开绑骨的图生3D资产是单一整体网格（用工具解析确认过：车辆模型只有1个mesh节点、0个动画片段），车轮转不动，人物只能靠整体位移加上下起伏假装走路。

我说那句「肯定要做骨骼能动的游戏」之后，这条路径被单独跑了一遍，跑通了。

在Tripo这一侧

1 选模型，把骨架预设换掉

工作台的「动画」页，AI模型下拉默认是「v2.5 -适用于动物」，人物必须换成类人预设。这是第一个容易漏的地方。

2 点「自动绑定」，20积分

我第一次绑完中央预览是空白的、旁边出现「重试」按钮，以为失败了。刷新页面之后骨架预设会重置回「动物」，所以第二次得重新选一遍类人再绑。第二次成功。这一步我花了40积分，其中20是白花的。

3 在动画库里搜「行走」

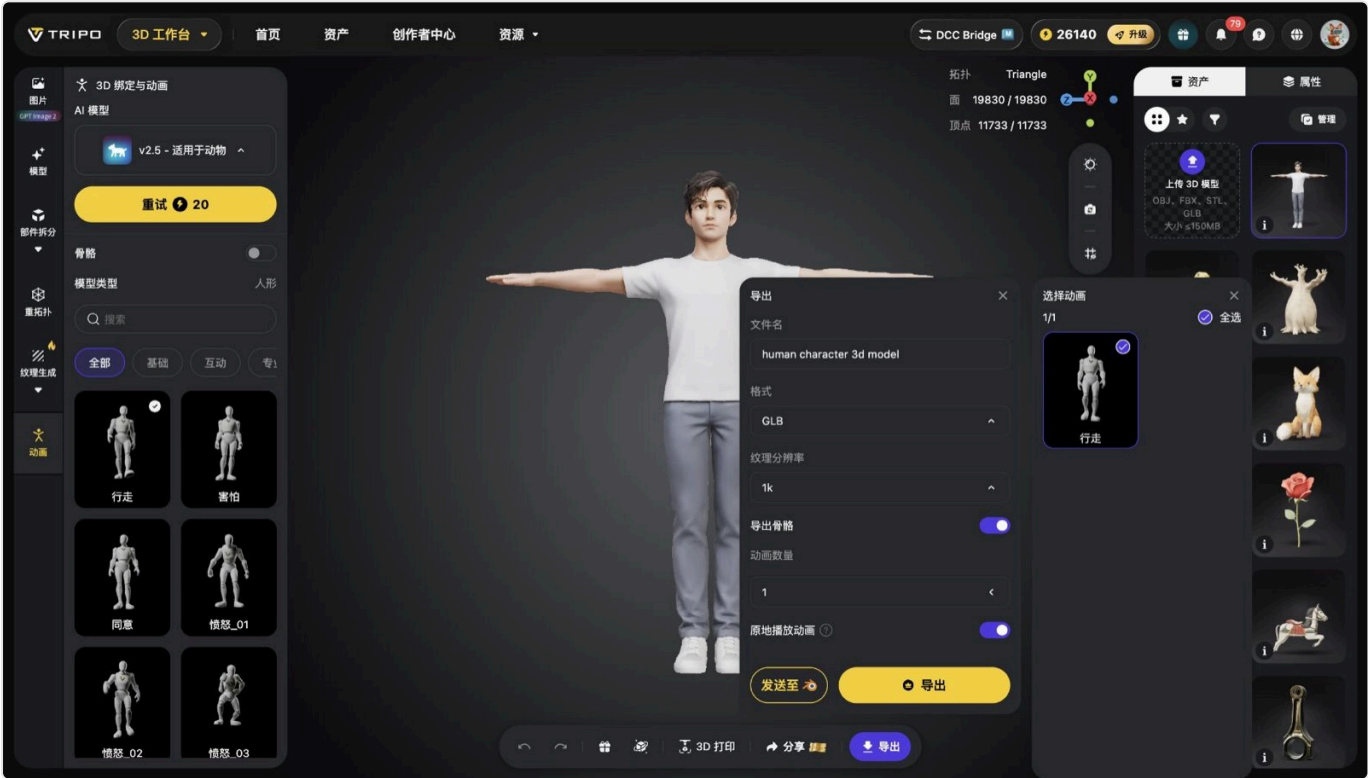
类人骨架的动画库有近一百个预设，分基础 / 互动 / 场景 / 情绪几类，有搜索框。

4 在导出弹层里勾上它，这一步最容易漏

点动画卡片只是预览，不等于应用。必须在导出弹层的「动画数量」下拉里勾选，数字从0变成1才算带上。我第一次导出拿到的就是一个有骨骼、没动画的GLB。

5 打开「原地播放动画」

勾上之后动画只管迈腿，位移交给游戏代码管，这正是游戏角色需要的形态，不勾的话模型会自己往前走，和引擎的位置计算打架。



整个绑骨流程里最关键的一屏：右边「选择动画」里「行走」已勾选，中间导出弹层的「动画数量」显示1、「原地播放动画」开关是打开的。这两个地方任意一个没设对，拿到的都是一个不会动的骨架。2026-09-10。

三个版本的对照，能看清每一步到底加了什么：

版本	skins	关节	动画	体积
原始静态 (Tripo直出)	0	0	0	443 KB
第一次导出 (漏勾动画)	1	41	0	1309 KB

版本	skins	关节	动画	体积
最终版（绑骨+行走）	1	41	1	1352 KB → 压缩后584 KB

那个行走动画拆开看：123个通道，其中16个带关键帧（真正在动的腿、臂、脊柱，最多57帧、最少51帧），采样间隔0.042秒（约24 fps），时长2.375秒一个完整走路循环；剩下107个是保持不动的关节（手指之类），这是正常结构。

在引擎这一侧，改了三处

一，实例化必须走 `SkeletonUtils.clone()`。带骨骼的模型不能用普通的 `clone()`，否则所有实例共享同一套骨骼，一街的行人会做一模一样的动作。

```
import {clone as skeletonClone} from 'three/addons/utils/SkeletonUtils.js';

const root = skeletonClone(walkScene);
const mixer = new THREE.AnimationMixer(root);
mixer.clipAction(walkClip).play();
```

二，每个行人一个自己的 `AnimationMixer`，每帧调 `mixer.update(dt)`。

三，相位要错开。用行人自己的坐标算一个偏移量喂给 `setTime()`：

```
mixer.setTime((x * 0.37 + z * 0.11) % walkClip.duration);
```

不错开的话，一排人会像仪仗队一样齐步走。这个细节没人会告诉你，但少了它整条街都是假的。

还有一个坑值得单独记：`three/addons/utils/SkeletonUtils.js` 在项目的 `vendor` 目录里根本不存在。我为了不依赖CDN，`vendor`里只精选了 `GLTFLoader`、`OrbitControls`、`BufferGeometryUtils`三个，`import`一个不存在的文件会让整个游戏白屏。补一个对应版本的文件进去才通。



验证真的在走：同一个行人相隔0.7秒的两帧，手臂摆幅、腿的姿态和位置都变了。判断标准不能是「看起来在动」，要能指出哪个关节动了。2026-09-10。

这一步走通的时候，场景里18个行人全部挂上了那具骨骼和那段行走动画。整条路走通花了两次20积分。唯一不满意的是只测了「行走」这一个动画，近一百个预设里还有跑步、挥手、打电话，质量稳不稳定我没验过，这条得说清楚。

注意

「18个行人全部用上了真动画」这句话我当时写进了开发笔记，后来发现它同时也是一个bug的描述。为什么，看下面第四个根因。

梧桐的注入时机

把 § 17 那两棵梧桐接进城市层，技术上只有三十几行改动，但有一个时序约束必须先说：城市层是在build的时候就建好实例池的，等它建完再给树就来不及了。所以加载顺序是死的：

```
window.__PARIS_TREE_GLB = await loadCityTree();    // 必须在前
await buildReferenceCity(...);                    // 城市层 build 时读它
```

其余三处小改都在城市层副本的家具模块里：近景树冠池的几何和材质改成读注入值；填树函数的近景分支分两条路径算实例变换；GLB树自带树干，所以不再叠画程序化树干。集成之后的性能账里有一条反直觉的：

指标	改前	改后
绘制调用	237	231 (少了6个)

指标	改前	改后
三角面	2,340,888	2,842,088 (+21%)
帧时间中位数	8.40 ms	16.70 ms

绘制调用反而少了，因为近景池从四套程序化变体合并成了同一个GLB，白赚。帧时间看着翻倍，其实是从119 fps掉到60 fps的垂直同步上限，原本那点余量被吃掉了，60 fps本身是满帧流畅的。这种数字要看清楚再下结论，不然会把「用光了余量」误读成「性能塌了」。

注意

这本书里的性能数字一共有三组，它们互相不能比，先把口径说清楚。

- 上面这组（237 / 231调用，284万三角，16.70 ms）：2026-09-10，桌面高画质，树接入前后的*同一机位对照*，帧时间是浏览器的帧间隔，被垂直同步锁在8.3或16.7毫秒。
- § 19那组（202调用，248万三角）：同一天的三套QA巡检，相机是跑在赛道上采样的，**不是上面那个机位**，所以调用数和三角面都对不上，这是正常的。
- 后面第三轮修复那组（77调用，144万→244万三角，2.7→5.0 ms）：2026-09-12，代码已经改过好几轮，量的是**渲染耗时**不是帧间隔，机位是街上一个静止点。

三组数的日期、机位、口径三样全不一样，并排看必然打架。我没有把它们统一成一组，因为那要重跑全部测量，而重跑之后代码又变了。诚实的做法是每组都标清条件，读者自己知道该拿哪一组去和自己的项目比。

这件事我后来又撞了一次：换完资产在街上机位量到146个绘制调用，和上面说的77对不上。查下来两个原因都排除不掉，一是那次基线没记相机坐标、我复现不了同一个机位，二是那半天里另外两条线也在改代码。所以那个读数只能当「新资产下的现场读数」，不能当「和基线的严格对比」。性能数字的第一要求不是准，是能说清它是在什么条件下量的。

三轮修复：四条反馈，七个根因

上面那些是把东西装起来。装起来之后我自己开着玩了几轮，报了四条反馈，一条比一条口语：「树会突然刷新和消失」「碰撞机制需要做得更好，还是很容易穿模」「巴黎街边哪有这么奇怪的植物。。。莫名其妙啊」「试试撞车，体验下异常的效果」。

这四句话追下去，挖出七个根因，没有一个的根因是「机器不够快」。我把它完整写在这里，因为这是这本书里最像真实工程现场的一段：看见的症状、猜错的方向、最后量出来的数。

根因一：抢不到坑位就整棵不画

「树会突然刷新和消失」，我第一反应是GPU顶不住了，准备去砍面数。结果一条都不用砍。
近景的树是放在一个实例池里画的，池子有容量上限，代码大概是这个意思：

```
if (距离 < 近景半径){
  if (这个变体的坑位已满) ok = false;    // ← 落到这里就什么都不画
  else { ...放进近景池 ... }
} else if (中景池还有空){ ...放中景冠 ... } // ← 这个 else 够不着
```

问题全在那行注释上。一棵树只要落在近景半径里、又没抢到坑位，它就什么都不画，而不是退一步画成简化的中景树冠。那个能降级的分支挂在 `else` 上，近处的树根本走不到。

后果是反直觉的：**165米圈内抢不到坑位的树凭空消失，而200米外的树好好画着**。那条大道上165米内实测有96株候选，旧配额只给36个坑，也就是有60株是看不见的。

加上另外两件事，「突然刷新」也解释了：近景配额是**按变体均分**的，36棵被拆成两个18，这一段街要是树种偏向某一个变体，那个变体先占满、后面的抢不到，而另一个变体的坑位还空着；再加上重排的触发条件是相机每走满80米重填一次，开到60 km/h就是每5秒整批换一次。**不是渐变，是定时集体重排**。

改法	改成什么
近景配额	按变体均分改成 全局计数 ，谁近谁先占
抢不到坑位	降级成中景冠 ，不再整棵不画
重填间隔	80米 → 12米 ，并且树单独一个订阅，不再和路灯长椅共用
近景上限	36 → 110 （流畅档44）

改完在浏览器里复核，不是推算：同一机位，近景GLB树的距离分布是0到164.5米，中景程序化冠从165.1米起，**中间没有断档**。沿大道每12米采一次，240米里每步只换0到14棵，多数步是0；旧版是每80米整批36棵重排。

根因二：同一个bug还在另外两处

修完树，我让agent做了一次全量体检：全城60个采样点，把每个实例池的占用率打出来。37个池里有11个跑满。跑满本身不是bug，关键看溢出的时候丢的是远处还是近处。查下来又找到两处和树同源的。

路灯和长椅：半径被配额锁死。池子一旦跑满，LOD的实际半径就由配额决定，而不是由距离环决定。路灯那一环名义上是1100米，实际被100盏的上限锁在约300米；而那个由配额决定的边界又跟着相机每80米重排一次，于是一批批地进出。把上限从100提到420之后，实测有效半径从300米变成629米，已经在雾里了，看不见了。

奥斯曼屋顶的烟囱：和树一模一样。烟囱分两个池，近池620个完整烟囱（每个116三角），远池是剪影（每个10三角）。代码同样是「近了就放近池，否则放远池」，而近池满了推入会失败，**500米以内抢不到那620个坑的烟囱整根消失，600米外的反倒画着剪影**。实测近池在密集街区275米就填满了。修法是一行：近池满了退到远池的剪影，不是不画。

核心建议

这个修法有个天然正确的地方值得指出来：**池子是由近及远填的**，所以退到远池的永远是「更近的先占坑」，被丢掉的仍然是最远的。降级顺序不需要额外排序，它自己就对。

根因三：配额调高几乎免费

三处配额全部调高之后的总账，是这一整节我最想让你看见的一张表：

	改前	改后
街上机位中位渲染耗时	2.7 ms（近景树36棵）	5.0 ms
三角面	144万	244万
绘制调用	77	77（一个没增）
折算帧率	—	约200 fps

多出来的100万三角面全部走既有的实例池，**绘制调用一个没加**。这是这套架构最值钱的地方。

这套架构里调高配额几乎免费，而之前所有的「闪」都是为了省这笔根本不需要省的钱。

这句话我想请你在自己的项目里念一遍。实例池这套东西的成本结构是：提交一次绘制调用很贵，往里多塞几百个实例几乎不要钱。所以「怕卡所以把上限压低」这个直觉，在这套架构里是**反的**，它省不下多少，却会让近处的东西凭空消失。

根因四：那棵树根本不像树

把近景配额从36提到110之后，我在游戏里截了一张图发过去：「巴黎街边哪有这么奇怪的植物。。。莫名其妙啊」。

我是对的，但这不是agent改坏的。它把一直就有的**问题放大了**。

把那个树模型单独渲出来看（不是在游戏里找角度，是拿Blender出六视图），问题一眼就清楚了：**7,473个三角面撑不起一个树冠**，出来是几十片巨大的独立叶片挂在细枝上，像一棵幼苗或者某种奇怪的室内植物，完全不是巴黎那种被反复修剪、树冠密实的行道树。配额36的时候一屏里只有几棵，混在程序化冠里看不太出来；提到110之后满街都是它，一眼就假。



形体问题只有单独渲出来才看得见。上排是旧的 `tree-01-plane-green`，7,473面，几十片巨叶挂在细枝上；下排是换上去的P2.0七叶树，11,825面，树冠密实、分枝结构真实。六视图，Blender离线渲染，不是游戏内截图，因为游戏里总能找到一个角度让它看起来还行。2026-09-12。

注意

教训：三角面预算是按「够不够画」定的，没人按「够不够像」验过。这两件事需要两种完全不同的检查。前者看性能面板，后者只能把模型单独渲出来、一件一件看。我这次还是先改了配额才想起来去看模型。

换件之后的账：

	旧（稀疏叶片）	新（七叶树）
三角面	7,473 / 7,913	11,825
贴图	512 ² ，3张	8192 ² 单张，降到1024 ²
文件	0.39 / 0.45 MB	15.7 MB → 1.01 / 1.06 MB
形体	稀疏叶片挂细枝	密实树冠

担心的是面数涨了会不会把刚修好的余量吃掉。实测把两个树池的数量依次砍到八成、六成、三成五，中位渲染耗时是3.1、4.1、4.0、3.3毫秒，非单调。非单调的意思是树的数量在这个机位根本不是主导开销，四个读数全在噪声里。所以110这个配额不用回收。

顺带把株距也放宽了。原来的间距是照巴黎实际行道树写的，但从车里看出去是一道连续的绿墙；放宽之后读起来才是「一排一棵一棵的树」。全城树候选从2.27万降到10,765，同一机位近景树96棵降到65棵，三角面321万降到247万，帧时间6.6毫秒降到5.8毫秒。画面更像巴黎，机器还更轻松，这种改动不多见。

根因五：拿一个1.1米的圆去代表一台3.57米的车

「碰撞机制需要做得更好，还是很容易穿模」这条，根因藏在一个参数里：碰撞检测一直是拿半径1.1米的圆去代表玩家那台车。

而2CV实测是1.58米宽、3.57米长、1.62米高，半宽0.79、半长1.79。一个1.1的圆：

- 横着比车宽（0.79变成1.1），贴边过窄巷会被莫名其妙挡住
- 纵着远比车短（1.79变成1.1），车心还没碰到墙，车头已经插进去0.7米

这就是「很容易穿模」的全部。改法三条：

1 前后双探针

车心前后各偏1.0米、各带0.85米半径，拼出来的胶囊是3.70米×1.70米，和车身几乎一样。两个探针各自求解，两份修正量叠加作用在车心上。单独推某一端会让车绕着另一端转，看起来像被墙吸住。

2 用上「上一帧在哪」这两个参数

原来一旦点进到多边形内部，代码把它推向最近的边，而深入过半之后最近的边是对面那条，于是车被顶穿到墙的另一侧。上一帧的坐标一直传进来却从没被读过。现在先沿来路二分退回接触点，再做边缘推出，推出方向必然朝外，没有歧义。

3 兜底闸

两趟迭代之后如果还在实心块里，直接退回上一帧。最坏情况是车停住，不会是车穿进去。

验收方法是把车全速怼进墙里，全城取61个撞墙点，二分量车头的插入深度：

	平均插入深度	零插入的样本	完全穿过
改前（单圆r=1.1）	1.10 m	—	有
改后（双探针+回退+兜底）	0.40 m	32 / 61	2 / 61

剩下那2个是同一个点，两个实心块叠着，而且那个测试起点本身就在实心块内部，兜底闸要求「上一帧是安全的」才触发，所以没兜住。游戏里车不会凭空出现在建筑里，这个状态到不了。但我也没能证伪，如实记着。高速（每帧0.57米，约122 km/h）和中速（每帧0.35米，约76 km/h）两档结果一致，没有隧穿。

根因六：码表永远显示2 km/h，是一个不动点

「试试撞车，体验下异常的效果」。前台截图抓不到撞墙那一瞬间（标签页不在前台的时候，浏览器的动画循环是停的），所以改成把撞墙那几十帧逐帧打出来。异常一眼就看见了：

```
f222  v=23.00  位移=0.383  ← 撞击前
f223  v=14.95  位移=0.361  ← 撞上
f224  v=9.92   位移=0
f240  v=0.60   位移=0
f248  v=0.59   位移=0      ← 永远停在 0.59，不会归零
```

位移确实是0，车被挡住了，但速度永远降不到0，稳定在0.59 m/s。

这不是玄学，是一行代码的不动点。碰撞的处理是 `speed *= .65`，而每帧油门加 `19 * dt = 0.317`，解方程 $v = (v + 0.317) \times 0.65$ ，得0.589。它就停在那儿了。

三个后果：码表贴着墙一直显示约2 km/h，车明明没动；撞墙没有任何回弹或反馈，是「贴住」不是「撞到」；那点残留速度会在打方向的瞬间释放出去。

改法是不再无脑乘一个系数，而是按墙的法线把速度拆成两份：撞进墙里的那一份（法向）去掉，并且多去15%，那15%就是回弹；沿着墙的那一份（切向）留86%，那就是摩擦。

```
const vn = vx*nx + vz*nz;          // 法向分量，撞进去时为负
if (vn < 0){
    vx -= nx*vn*1.15; vz -= nz*vn*1.15; // 去掉法向 + 15% 回弹
    vx *= .86; vz *= .86;              // 切向摩擦
}
```

改后正面撞墙是23 m/s在一帧内卸到2.97，之后在0.05到1.5之间小幅抖动（油门顶着墙、墙推回来），位移0.006到0.029米，贴着墙轻微蹭动，不再是个死值。撞完打方向，速度从0.46平滑爬到1.27、位移持续增加，车顺着墙滑出去了。这是改之前完全没有的行为，「贴着墙过弯」这种开法第一次成立。



撞墙现场。这一帧是逐帧打印时同步截下来的，车贴在墙上、提示条弹出来「撞上了。松油门、回一点方向再走」。撞击的视觉反馈到现在也还没做：不抖屏、不掉漆、没有粒子，这一轮只把物理改对了。2026-09-12。

根因七：街上18个行人其实是同一个人

这条是并行的另一条线在做资产替换时发现的：它换了两个静态行人的槽位，却发现画面里根本不出现它们。

根因在生成行人的那个函数 `makeWalker(key, height, x, z, rotation)` 上。它第一行就是「如果绑骨行人加载成功就走这一支」，而绑骨行人当然加载成功了，所以传进来的 `key` 从头到尾没有被用过一次。结果街上每一个行人都是同一个藏青西装、戴礼帽、拎公文包的男人。另外五件静态行人资产的处境更微妙：它们被加载进了显存，但一个实例都没有画出来。

注意

这个bug有个特别值得学的性质：**它不报错、不掉帧、不白屏，三套自动化QA也全绿**。因为从程序的角度看，18个行人确实都正常创建、正常动画、正常渲染了。要发现它，只能是人抬头看一眼街上，或者像那条并行线一样，换了东西之后去确认「我换的东西真的出现了吗」。**改完东西要去画面里确认它出现了，这一步不能省。**

改法：三个人里挑一个用绑骨那具会走的（全场只有这一具有骨骼和行走循环），其余用各自的静态模型，五种轮换。所以前面那句「18个行人全部用上了真动画」，正确的说法是「18个行人里有6个在真的走，其余12个是五种不同的人站着」，这比满街同一个人可信得多。

顺带修掉第二个假点：**静态行人不再左右滑动**。原来所有行人都吃一个左右各3米的正弦摆动，配合走路循环是对的，配在一个静止的模型上就是「人在地上滑」。现在不走路的那些，摆动量直接给0，站着不动。

七个根因的共同点

回头看，它们长得完全不一样：LOD配额、碰撞体形状、速度衰减的不动点、一个从没被读过的参数、一件形体不对的资产。但它们有两个共同点，我觉得比每一条的修法更值得记：

一，全都伪装成了别的问题。树的问题看着像显卡不够，实际是逻辑；撞墙的问题看着像玄学，实际是一行乘法的不动点；行人的问题根本没表现成问题，它表现成「一切正常」。**如果我按症状去优化，会去砍面数、去调物理参数、去给行人加更多变体，几个方向全是错的。**

二，那四条反馈全是我开着车玩出来的，没有一条是自动化验收发现的。这不是说自动化验收没用，下一节会讲它抓到了什么。而是说这两类检查的覆盖面根本不重叠：**程序验证「它有没有按代码跑」，人验证「它像不像那么回事」**。后者目前没有任何办法可以外包。

两个卡了我半天的坑

坑一：追尾镜头方向算反了。镜头一度卡在车前面而不是车后面，开起来是在倒着看自己。这类错误没有报错，只有「感觉不对」，定位全靠把符号一个个试过去。

坑二：沙盒环境禁用 `fetch()`，模型渲染成纯黑色，没有任何报错。

这一条值得展开，因为它是个通用坑。Three.js的 `GLTFLoader` 加载贴图时会优先走 `ImageBitmapLoader`，而它内部用的是 `fetch()`。某些沙盒预览环境会完全禁用 `fetch()`（哪怕是同源的 `data: URI`），于是贴图静默加载失败，模型全黑，控制台干干净净。

解法是一行：

```
window.createImageBitmap = undefined; // 逼 Three.js 退回走 <img> 标签的经典路径
```

更根本的一条是：**多文件加载还是base64内联，由发布平台决定，不是技术偏好**。早期版本因为锁定在那种沙盒里，只能把模型全转成base64塞进单个HTML，3个模型就吃掉12.5 MB，23个资产根本塞不下。一旦改成普通静态网页，真实HTTP环境没有这个限制，直接用相对路径加载 `.glb` 就行，问题从「怎么塞进16 MB」变成「怎么让加载别太慢」，完全是另一个问题。先问清楚要发到哪，再决定用哪种加载方式。



桥面高度是单独处理的：不能直接套街面 $y=0$ ，要按实际桥面网格取样再缓存。24座桥实测的桥面高度在0.39到1.38之间，套错了车会嵌进桥面里。2026-09-10。



按Tab打开的全城地图，右侧是8处地标的快速到达入口。地图数据源自OpenStreetMap，城市为艺术化改编，这行字在页面左下角，不能省。2026-09-10。

这一节的产出

一个能在本地HTTP服务上跑起来的完整游戏。但「能跑起来」离「能发出去」还有一整套验收：三套QA、一轮独立质控、一个离线可用的发布包。下一节讲那个。

§19 验收与发布

QA and Release

「能跑起来」和「能发出去」之间隔着三套QA、一轮独立质控和一个离线可用的发布包。这一节给你我实际用的验收清单、独立质控挑出来的三个真问题，发布到B站Toy那条路上的几个硬约束，以及不走Toy时那条更省事的静态托管路。顺便把一句不能当结论的话说清楚。

三套QA，各盯一件事

我跑了三套自动化验收，各自的职责分得很开：

文件	盯什么	关键结果
qa/city-results.json	玩法完整性	两圈竞速184.50秒完赛、单圈计时92.33秒、幽灵回放正常、16个检查点顺序正确、碰撞0次、8处地标到达后都能继续开、暂停与失焦正常、照片能下载、零外部网络请求
qa/mobile-city-results.json	移动端可玩性	390×844、375×667、844×390三种尺寸；真实触屏事件的多指转向+油门+松手；地图与拍照可用
qa/package-results.json	发布包能不能独立活	从发布包的子路径启动、屏蔽外网之后仍然加载成功、六个城市模块齐全、23件本地资产正常

第一套里有一项特别值得学：24座桥的桥面高度逐座采样，记下每一座的坐标和实际高度（0.39到1.38之间）。车的落地高度要是直接套街面y=0，过桥的时候就会嵌进桥面里。凡是「代码算出来的地形」和「玩家实际会走过去」有交集的地方，都要逐点采样，不能抽查。



竞速结算页：完赛用时、风格积分、漂移时长、金币数、碰撞与救援次数都在。QA会一路真实点击到这一屏，而不是直接调函数判定完赛。2026-09-10。

第二套的口径我写得很硬：**没有把模拟器结果当成实体手机性能**。桌面高画质下是202个绘制调用、248万三角面；移动端低画质是82个绘制调用、91万三角面，两边帧时间中位数都锁在16.7毫秒，但那是无头浏览器的帧间隔，不等于GPU耗时，也不等于任何一台真实手机上的帧率。这条边界不划清楚，后面就会有人拿着这个数字去承诺性能。



390×844竖屏：油门、刹车、方向、漂移、冲刺五个触屏按钮，多指同时按（比如左转+油门）实测两个输入都为true。2026-09-10。



844×390横屏：控件会重新排布到两侧，HUD压扁。三种尺寸各测一遍，因为横屏下最容易出现控件被安全区裁掉的问题。2026-09-10。



竖屏下的全城地图：地图和地标列表改成上下排布。地图能打开还不够，QA要验证「打开之后能关回去、关回去之后还能继续开」。
2026-09-10。

独立质控：写它的AI不审它

三套QA是主线程自己跑的，所以还要有一道独立的。**我的规矩是写内容的AI不审自己的产出**。另开一个只读游戏和参考项目、完全不看开发过程的agent，专职找问题。它交回来的那份报告，比我预期的狠。

两轮审查挑出三个真问题，每一个都值得抄进你自己的清单：

- 一、着色器在当前版本编译不过。**参考项目的立面材质钩子替换了全部UV声明、只声明 `vUv`，而当前r160的贴图片段读的是 `vMapUv`。它没有靠读代码下判断，而是开了一个无头页面把那段逻辑逐字跑了一遍，拿到真实编译结果：`runnable = false`，`ERROR: 'vMapUv': undeclared identifier`。
- 二、18个行人的y坐标每帧都是NaN。**创建行人时只记录了坐标和朝向，更新时却引用了一个不存在的字段，算出NaN直接写进了位置。**页面零报错、着色器零错误，但这18个对象根本没有有效的世界位置。**它的验证方法是遍历场景、筛出位置分量非有限值的对象，正好18个。
- 三、地标快速到达的落点偏了161.76米。**双层桥那一处按钮点下去，车落在离目标161.76米的地方，也没收到印章。把桥段加进道路查询之后重测，落点距目标0.0198米。

第二条是我最想让你记住的。它推翻的是一个所有人都在用的验收习惯：「**页面能打开、控制台没报错**」什么都不证明。独立质控自己在报告末尾写了一节叫「检查方法本身的盲区」，三条：

- 零 `pageerror` 不代表对象坐标有效，行人NaN就是反例
- 为了加速而关掉渲染的那次键盘测试，它的16毫秒帧时不能当性能证据
- 赛道中心线畅通、两圈完赛，推不出1375个街廓和全部桥头都有正确碰撞

核心建议

让审查agent在报告里单独写一节「我这次的检查方法覆盖不到什么」。这一节的价值经常比问题清单本身还高，它告诉你下一轮该往哪儿测，也拦住了你把「这轮没测出问题」读成「没问题」。

两条会把结论验反的教训

上面讲的是「验了什么」。这一段讲更要命的一层：验法本身错了，你会拿到一个看起来很扎实、方向却完全相反的结论。我在同一个项目里栽了两次，两次都差一点按着假结论去改代码。

一、手感类改动只在简化赛道上验，会把结论验反

§ 18那个「高速转向要收敛」的改动，第一版给的抓地系数比定稿激进得多（23 m/s下是0.62，定稿是0.71）。在自动化验收自带的那条546米简化赛道上，它的表现好得没话说：单圈从46.7秒降到43.4秒，风格分翻了2.4倍。

差一点就这么定稿了。拦住这件事的只有一个念头：546米、弯道很缓的测试赛道，和2116米的真实城市赛道，真的是一回事吗。于是把旧引擎挂到同一个真实赛道上直接对跑，五档抓地系数各跑一遍：

23 m/s的抓地系数	城市赛道单圈	大偏航帧数	撞车
1.00（原版）	91.1 s	71	0
0.87	90.3 s	152	0
0.79	89.6 s	237	0
0.71（定稿）	87.8 s	413	0
0.62（第一版）	166.2 s	3001	0

0.71和0.62之间是一道断崖。0.62那档在城市赛道上全程推头、全程漂移，单圈从91秒崩到166秒，大偏航的帧数从71涨到3001。而0.71这档反而是全场最快的一档，比原版还快3.6%，撞车仍然是0。

注意

教训：手感类的改动只在简化测试赛道上验，会把结论验反。简化赛道存在的意义是「快速回归，看有没有跑崩」，它回答不了「好不好开」。凡是改的是手感、摩擦、惯性、阻尼这类东西，必须拿到真实场景里跑，而且要扫一排参数看有没有断崖，不能只测你想定稿的那一个值。

还有一条边界要说清楚：这一整轮没有人手真试驾过。自动验证器回答的是「能不能开完、快不快」，不是「好不好开」。上面那张表的真正用途是当调参的地图：往0.79走更保守，往0.66走更飘。

二、游戏循环每帧都在覆盖你的相机

为了确认八个地标是不是真的都长出来了，我让agent做了一个巡检脚本：八个地标逐个把相机放过去、朝它该看的方向看，各截一张图，回答「站在那儿到底能不能看见它该有的东西」。

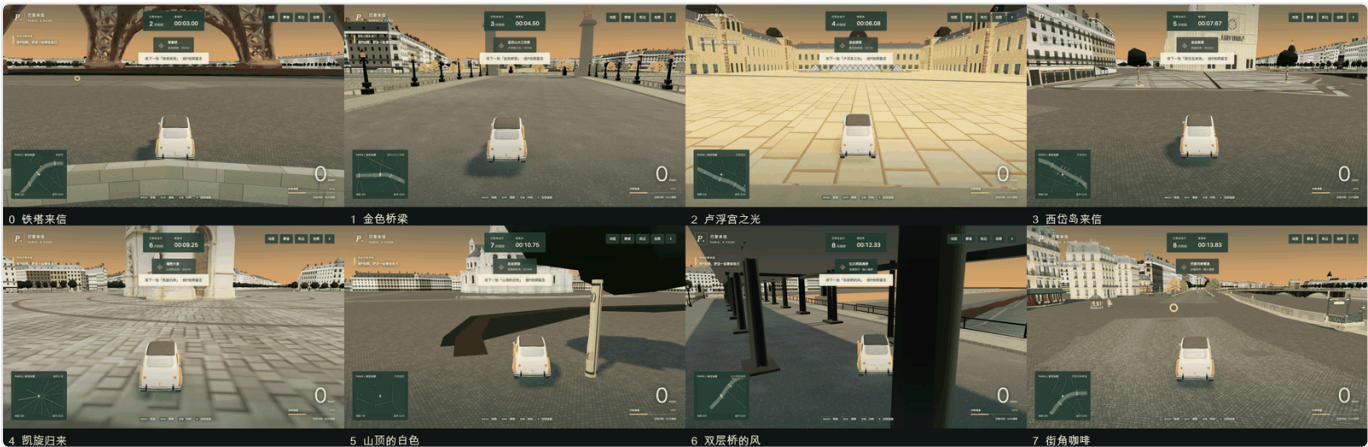
第一版结果很吓人：八个里七个看不见该看的。「金色桥梁」那一张没有桥，「岛上的钟楼」是一面白墙，卢浮宫是一片空地。差一点就去把新生成的那批地标资产往城里塞了，而那就是一整个下午的活。

拦住它的是一个不对劲的读数：第6个「双层桥的风」完美拍到了铁塔。如果真是资产缺失，不可能只有一个好的。

这种不一致本身就是「测法错了」的信号。七个坏一个，比八个全坏更值得怀疑。真实的缺陷很少这么挑人。

根因：这套demo的相机是游戏循环拥有的。页面在无头浏览器里是「可见」的，动画循环照跑，游戏每帧都用跟车相机把我设进去的那个相机覆盖掉。我截到的根本不是我摆的机位，是跟车相机在那个坐标上、朝着玩家当前车头方向的随机视角。

正解是不要去设相机，把车头转向地标，让跟车相机自己对过去，再等1.4秒让它插值到位。重做之后八个全部正常：站在铁塔腿下、桥上金柱在前、卢浮宫中庭看得见玻璃金字塔、凯旋门拱下、圣心堂白色穹顶、比尔阿凯姆双层钢桥、街角咖啡，一个不缺。



改用「转车头」之后重做的地标巡检，八宫格全部正常。第一版用「摆相机」的做法，这八张里有七张是空地和白墙。注意这里面混着两类地标：铁塔、凯旋门、圣心堂是Tripo生成的，卢浮宫和圣母院是代码画的，巡检不区分来源，它只回答「站在那儿看得见东西吗」。2026-09-12。

核心建议
记一条能直接搬走的：谁拥有那个状态，谁就每帧覆盖你。相机、时间、输入这些被主循环拥有的东西，你从外面设进去的值活不过一帧。要改的是驱动它的那个东西（这里是车），不是它本身。我同一天在这上面栽了两次，前面那次是想把车摆到墙边去截撞墙的画面。

顺着这条还有两个截图相关的坑，一起记在这：浏览器标签页不在前台的时候动画循环是停的，所以前台截不到撞墙那几帧，得改成逐帧打印数值；后台标签页的WebGL根本不渲染，截出来是纯黑，别把黑图当成「渲染坏了」。

还有一条在 § 18 已经讲过，但它和这两条是同一类，值得在验收清单里再写一遍：模型的形体对不对，必须把它单独渲出来看，不能靠游戏里某个角度的截图判断。游戏里总能找到一个角度让它看起来还行。

发布包怎么构建

本地开发和发布是两件事。这个项目的启动和构建各一条命令：

```
# 本地预览 (ES 模块和 GLB 必须走 HTTP)
python3 -m http.server 8765 --bind 127.0.0.1

# 构建发布包
python3 build.py
```

build.py 干的事很朴素：把入口文件和三个目录（assets、vendor、city-reference）复制进发布包/web/，顺手把来源清单和所有.md排除掉（发布包里不能带本机路径），再打一个ZIP，最后吐一份构建回执：

```
{
  "files": 52,
  "uncompressedBytes": 18170667,
  "zipBytes": 11935880,
  "entry": "web/index.html",
  "externalRuntimeDependencies": false
}
```

最后那一行 externalRuntimeDependencies: false 是这份回执里最重要的一个字段。Three.js固定在项目的vendor目录里，运行时不碰任何CDN，这是为了让这个包在断网、在任何子路径下、在别人的服务器上都一样能开。package-results.json 里那条「屏蔽外网之后仍然加载成功」就是在验它。

还有一条容易被忽略的：旧的构建产物不删，移进 _archive/ 带时间戳存着。构建脚本每次跑都会先归档上一版，这样任何时候都能拿出上一个能跑的包做对照。

这条好习惯后来咬了我一口

换完一批资产之后我重建了发布包，顺手查了一遍包里到底装了什么。build.py 里那个复制目录的调用没有排除 _archive，所以每一个被换下来的旧件都被整包带走了：11个旧版资产，约7.9 MB死重。玩家下载的包里，有将近三分之一是已经不用的东西。

同时还看见另一件事：中文的归档目录名在ZIP里是乱码，显示成一串问号方块。这是macOS的zip不写UTF-8标志位那个老问题，不是文件坏了，但用户解压出来看见乱码目录，第一反应一定是包有问题。

顺手把 __pycache__、.DS_Store 和源FBX也一起排除掉，账是这样的：

	改前	改后
文件数	63	52
解压体积	32.1 MB	24.9 MB
ZIP	21.1 MB	15.9 MB

核心建议

注意别把这组数和上面那份构建回执对上：回执那一份是09-10那一版，包里的资产还没换过，所以是17.3 MB解压、11.9 MB的ZIP。这一轮是09-12换完新资产之后的重建。巧的是修完文件数又回到52个，但它们不是同一批52个文件。

这条会一直复发，因为它是那条好习惯的必然副作用：每换一次资产，旧件进 `_archive`，包就胖一圈。所以修法不能是「记得手工清一下」，必须是把排除规则写进构建脚本，让它自己不去看那个目录。改完重跑发布包验收，仍然通过、仍然锁60 fps、零报错。

顺带一个只有中文用户会遇到的坑：

用Node跑构建或验证脚本时，`new URL(import.meta.url).pathname`遇到中文路径拿到的是一串百分号编码，`import` 直接报模块找不到。用 `fileURLToPath()` 转一下就好。我们的项目目录带中文，这个坑一定会撞上。



从发布包子路径启动的首页，屏蔽外网状态下加载成功。验收要在发布包上重跑一遍，而不是在开发目录上，两者的路径基准不一样。2026-09-10。这张里的铁塔还是换模前那一版，顶上是平的；§ 04、§ 12那几张是换模之后重拍的，塔尖和天线都在。同一个资产的前后两版，不是两件东西——随书资源包里放的也是这张图里的旧版。

发到B站Toy的三条硬约束

Toy支持纯静态的前端项目，把发布包目录或ZIP直接交给官方CLI就行。但有三条不看清楚就会翻车。

```
Available Commands:
  completion  Generate the autocompletion script for the specified shell
  create      创建 Toy 并提交审核
  help        Help about any command
  history     查看本地发布/更新记录
  login       服务端 OAuth 登录
  logout      登出并清除本地会话
  mylist      查看我的 Toy 列表
  stats       查看 Toy 的访问统计 (PV/UV 与 互动量)
  update      更新 Toy 并提交审核
  upgrade     升级工具到最新版本
  version     打印版本信息
  video       管理 Toy 与视频的绑定关系
  whoami      查看当前登录用户
```

发布用的Toy CLI，`toy --help` 里的子命令一览。整本书这一步我只跑到 `--help` 这一层，没有上传任何包体、没有提交审核——真正的发布留给你自己按。这里只截了命令表这一段，`toy create` 的完整参数抄在下面的代码块里，可以直接复制。2026-09-11。

1 发布是「预览 → 确认 → 提交审核」两段式

带包体发布时，CLI先上传打包、生成一个预览链接，默认不提交审核。提交审核要额外加确认参数。这是个闸门，不是多余步骤。先自己打开预览链接看一眼，确认没问题再提交。

2 页面跑在子路径下，绝对路径会白屏

Toy页面的地址是 `/toy/<slug>/` 这种子路径。任何以斜杠开头的绝对路径资源引用，包能传上去、审核也可能过，但打开是坏的（白屏或者404）。这个项目从一开始就全用相对路径，就是为了这一步。

3 双击 `index.html` 不算发布方式

ES模块和GLB都必须通过HTTP访问。本地用 `file://` 打开一定失败，那不是你的包坏了，是协议不对。验收的时候一定要起一个真实的HTTP服务。

这三条落到命令上就是几个参数。`toy create --help` 的输出抄在这儿，省得你自己装一遍才知道有哪些开关：

创建 Toy 并提交审核。

<path> 可以是目录、单个 HTML 文件或现成 zip。未传 --title / --slug 时，CLI 会从路径名自动推导；带包体的创建会先生成预览链接，确认后才提交审核。分类使用服务端枚举；图标支持 PNG/JPEG/WebP，大小不超过 512KB，宽高不超过 1024px。

Usage:

```
toy create <path> [flags]
```

Examples:

```
toy create ~/fire-emblem-web
toy create ~/fire-emblem-web --title "Fire Emblem Web" --category GAME --icon ./icon.png --yes
```

Flags:

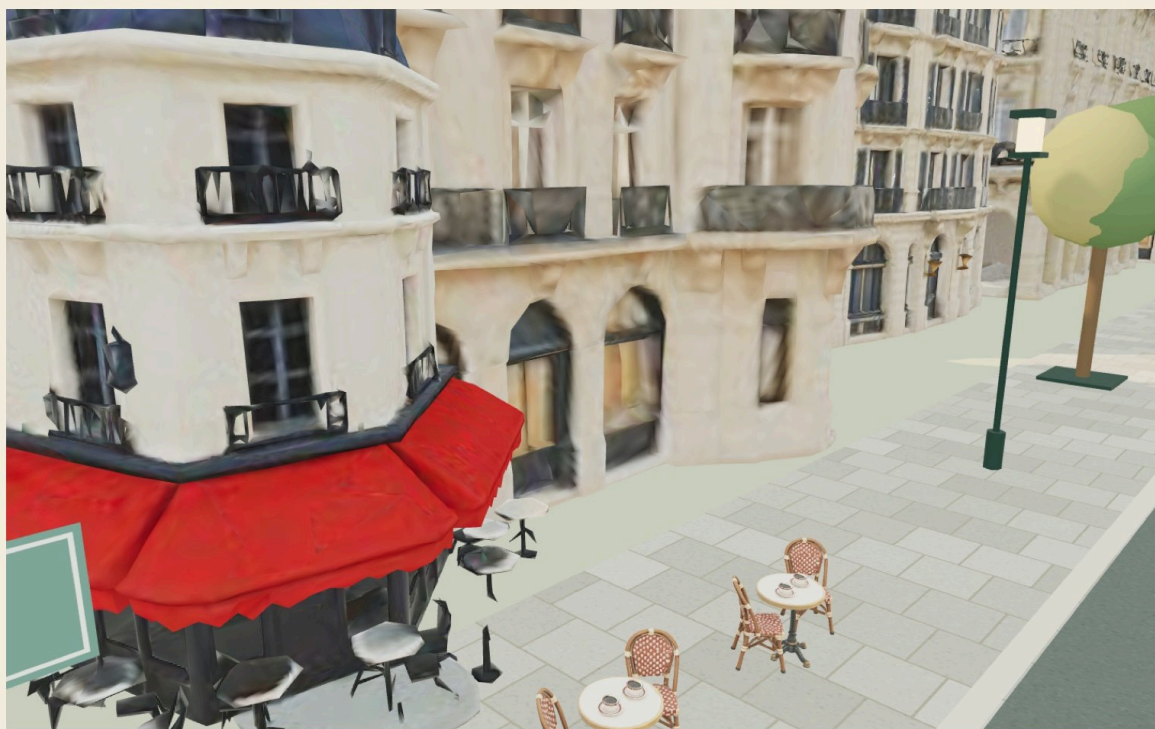
--access-password string	访问密码；创建密码访问 Toy 时使用
--category string	Toy 分类 (GAME INTERACTIVE_PLAY TOOL TEST GUIDE COLLECTION OTHER)
-h, --help	help for create
--icon string	Toy 图标图片路径 (PNG/JPEG/WebP, ≤512KB, 宽高≤1024px)
--poster string	封面图片路径
--slug string	Toy slug/sub_dir, 默认从路径推导
--title string	Toy 名称, 默认从路径推导
--visibility string	发布可见性 (link-only password public), 默认 link-only
--yes	预览后直接提交审核

Global Flags:

```
--json 读操作输出 JSON 而非文本
```

第1条里说的那个「确认参数」，就是最后一行的 `--yes`。

说明一下：到本书写作为止，这个demo还没有对外发布过。技术路径验证过可行，发布包也构建好了，但对外发布是一个需要人拍板的动作，不该由agent自己按下去。这条也写在了那张纸的第五栏里。

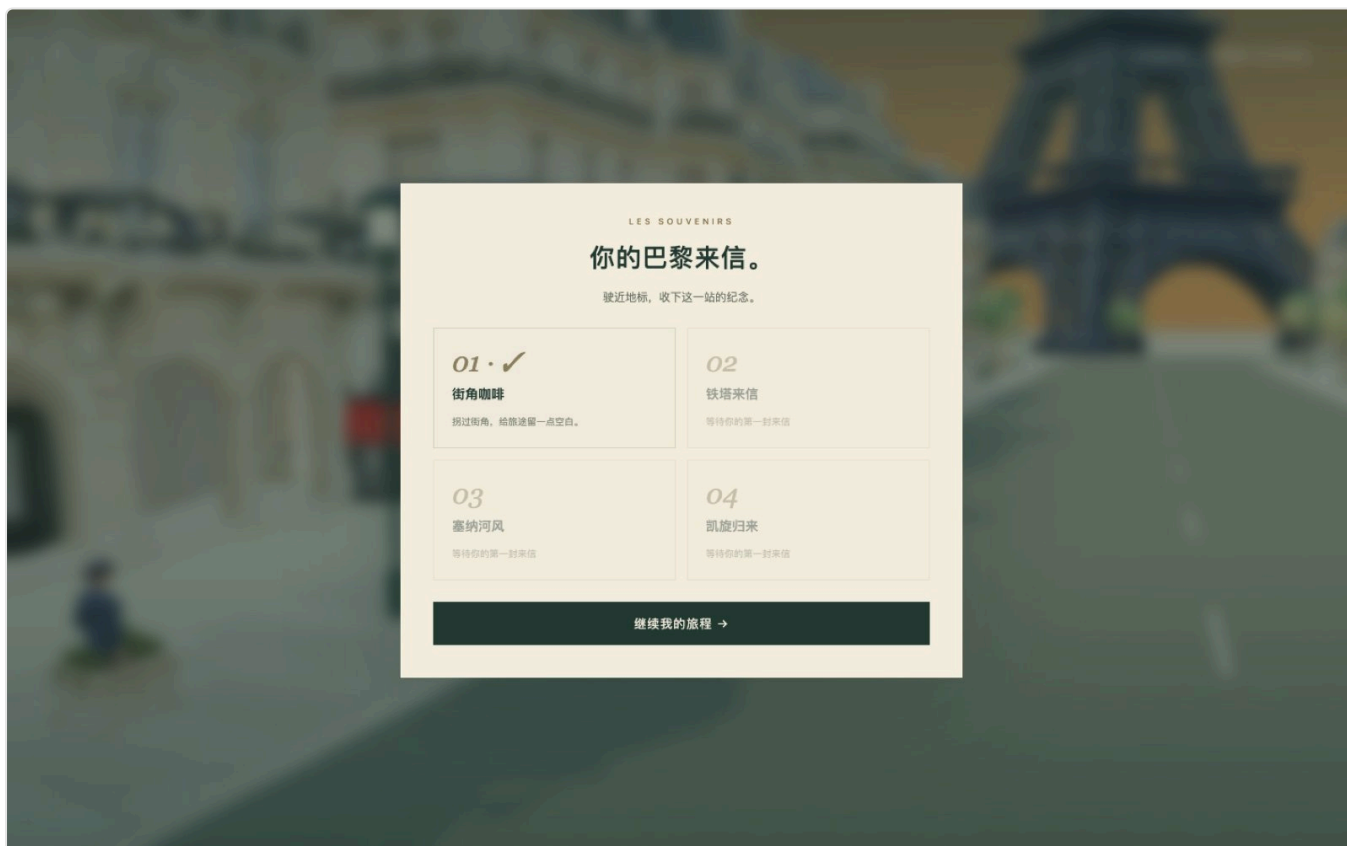


Paris, à fond.

巴黎来信 · 我的驾驶明信片

街角咖啡

拍照模式导出的静态PNG明信片。这张图里同时有两层：Tripo生成的咖啡馆门面与桌椅，程序化的人行道、路灯和远处那棵绿色球形树。QA里「照片能下载」验的就是这个产物。2026-09-10。



明信片册：驶近地标才能收下一张。进度存在本机浏览器里，换网址、换浏览器、开无痕都不互通，这句话得写进说明，不然玩家会以为丢了存档。2026-09-10。

不走Toy的话：扔到静态托管

Toy是B站生态里的那条路。如果你只是想做个东西发给朋友玩，还有一条更省事的：**静态托管**。§ 12里那句「Vercel、Cloudflare Pages、GitHub Pages都能直接扔」说的就是它——整个目录传上去，对方拿到一条链接就能开，不用等审核。

好消息是上面那三条硬约束里，只有第1条是Toy特有的。第2条「页面跑在子路径下」和第3条「必须走HTTP」在任何静态托管上都一样成立，所以自检动作是同一套。传上去之后再看白屏，比在本地先彩排一遍贵得多。

我拿资源包里的 `demo/巴黎/` 把这套彩排实跑了一遍，三条命令：

```
# 1. 扫绝对路径引用：这一条有输出，传上去就是白屏
grep -rnoE '(src|href|from)="/[^\"]*"' .

# 2. 把整个目录塞进一个两级子路径，模拟托管商给你的地址
mkdir -p /tmp/subpath/demo/paris && cp -R . /tmp/subpath/demo/paris/

# 3. 从上层起真实HTTP服务，浏览器开 /demo/paris/
cd /tmp/subpath && python3 -m http.server 8891 --bind 127.0.0.1
```

结果：第1条零输出；服务起来之后，入口页和 `app.mjs`、`engine.mjs`、`city-loader.mjs`、`city-map.mjs`、`style.css`、`vendor/three.module.js` 逐个取，全是200。这个目录一共52个文件、18170667字节，和前面那份构建回执的 `"files": 52`、`"uncompressedBytes": 18170667` 逐字对得上——资源包里放的就是09-10那一版发布包。2026-09-13实跑。

核心建议

第2步那个「塞进两级子路径」最容易被跳过，也最值钱。直接在目录根上起服务，所有引用都是对的，你什么都测不出来。必须让页面地址前面真的多出一段，才能把藏着的绝对路径逼出来。至于双击 `index.html`，那条连测都不用测：`file://` 下ES模块和GLB一定加载不上，§ 12已经栽过一次。

剩下的只有上传那一下。这一步我同样没按，理由和上一段是同一个：对外发布是要人拍板的动作。所以这条路上我能交给你的，是到「本地子路径彩排通过」为止的部分；再往后是三家托管各自的登录和上传流程，它们的文档里都是一条命令的事。彩排这一关过了，最常见的那类「传上去就白屏」不会再找上你。

那句不能当结论的话

项目一开始我给的目标里有两个数字：「提升1000%+的产品体验」「比海辛的案例出色100%以上」。

这两句是创作目标，不是实测结论。我没有统一量表，也没做过对照的用户测试，所以它不能出现在任何一句「我测出了什么」的表述里。项目README里我把这句单独写了一行，就是怕过几个月自己忘了。海辛的公开案例原帖在这里：

https://x.com/ring_hyacinth/status/2096704996381266424

这件事我想多说一句。做完一个东西之后，最诱人的时刻就是给它安一个倍数。但倍数是要有量表的，没有量表的倍数是情绪，不是结论。把「我当时想做到什么程度」和「我最后测出了什么」分成两句话说，是这一行最基本的体面。

已验证的和没验证的

推荐

已验证

完整两圈与单圈计时、幽灵回放、16个检查点顺序、8处地标到达后可驾驶、24座桥的桥面高度、真实键盘链路到结算、三种手机尺寸的触屏输入、照片下载、发布包在断网与子路径下启动、六个城市模块与23件本地资产

不推荐

没验证

实体手机的GPU性能（只跑过模拟器）、1375个街廓和全部桥头的碰撞（只测了赛道沿线、8个地标点和61个撞墙采样点）、行走之外的其他动画质量、有没有人手真试驾过（自动验证器只回答「能不能开完」）、1024贴图这一档没和512 / 2048做过A/B、新生成那批里没进城的那些在真实光照下的观感、对外发布之后的实际表现

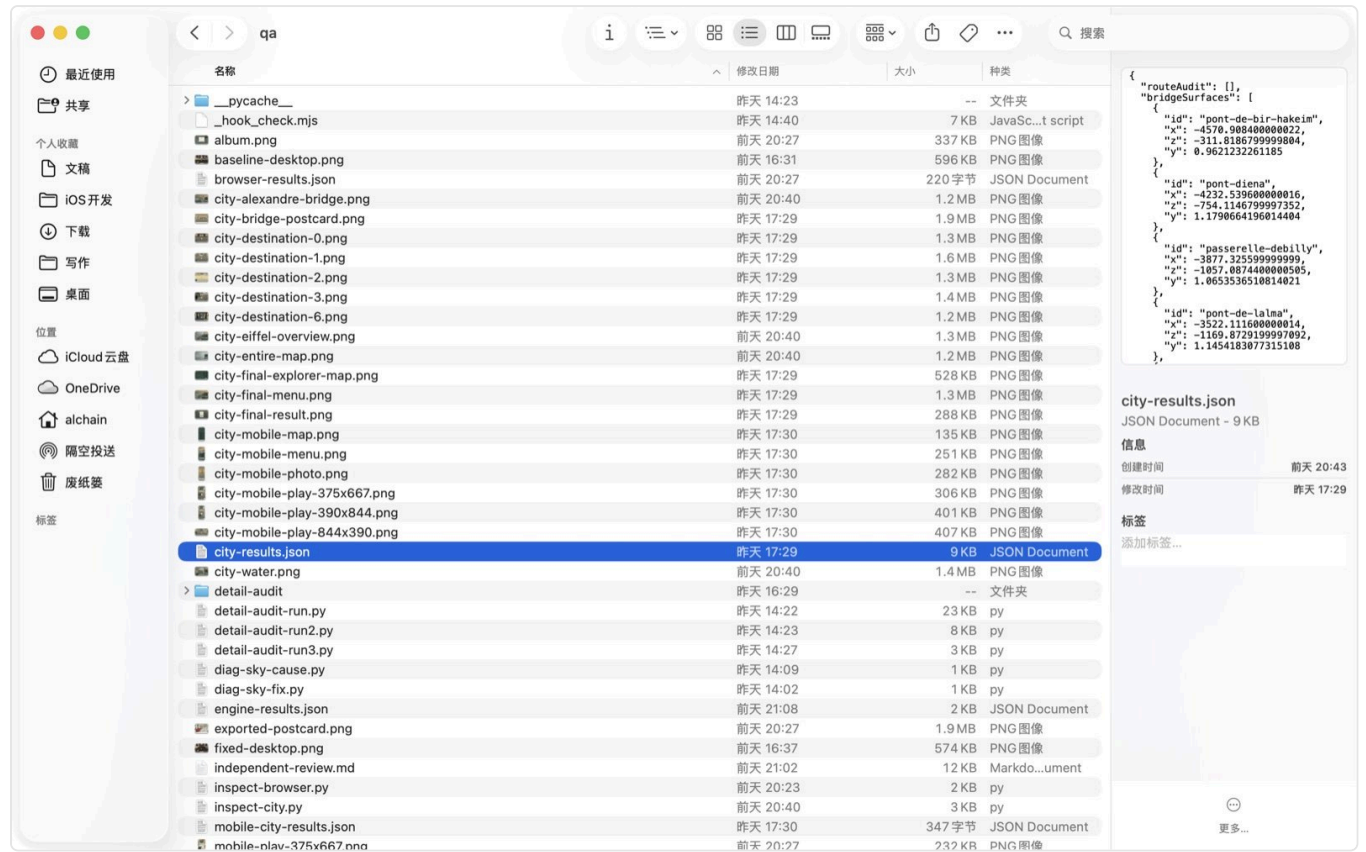
右边这一栏不是失败清单，是下一轮的工作清单。把它写出来，比含糊地说一句「测试通过」诚实得多，也有用得更多。而且这一栏每一轮都会变长，不会变短：你测得越多，知道自己没测什么的能力也越强。第一版的时候我根本写不出「没有人手试驾过」这一条，因为那时我还不知道自动验证器和真实手感是两件事。

Part IV到这里结束

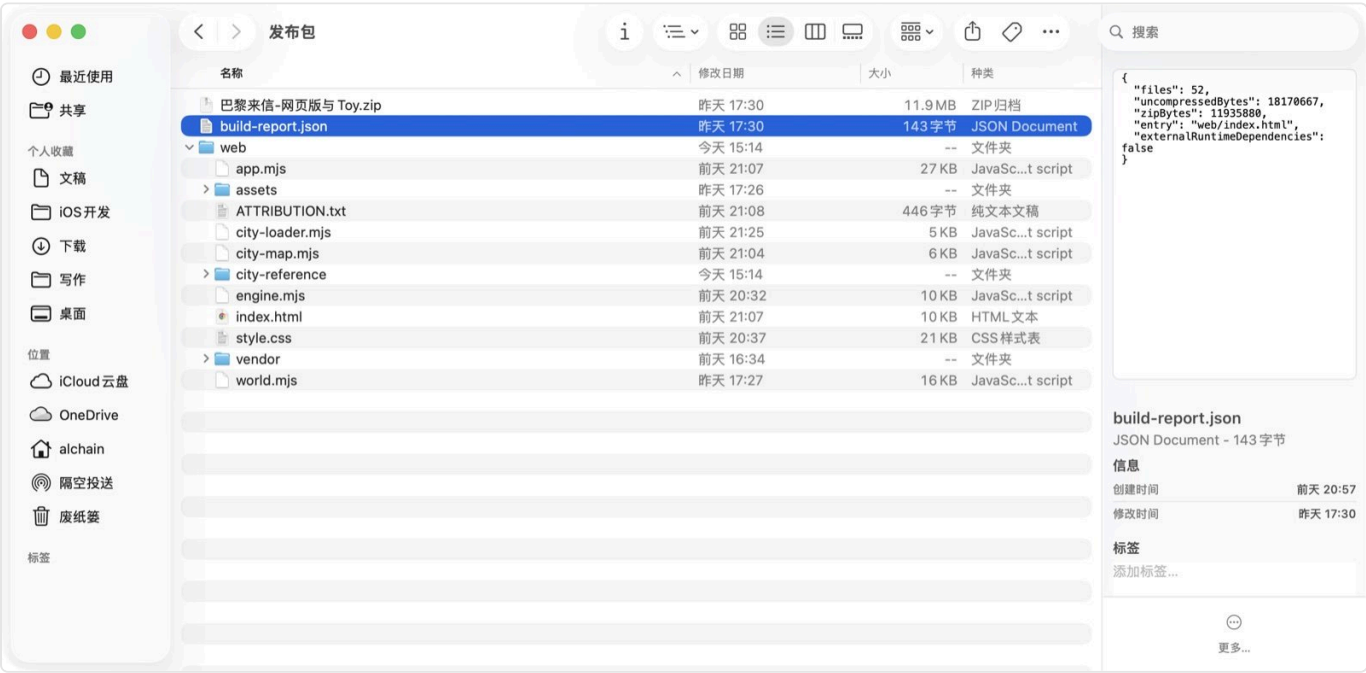
四节走完，你手上应该有一条完整的路：一张分工清楚的资产清单（§16）、一条能复用的生成与清洗流水线（§17）、一套模块边界清楚的组装方式和一份指挥agent的写法（§18）、三套验收和一个能独立活的发布包（§19）。

下一部分换个题目：同样这套方法，怎么用来做教学演示——一台能拆开的四冲程发动机、一幅能展开的油画、一颗能走进去的星球。

两张收尾的图



paris-demo/qa/ 目录，右侧预览栏是 city-results.json 的真实内容（能看到 bridgeSurfaces 里四座桥的实测坐标）。目录里其实有五份回执：browser、city、engine、mobile-city、package，上面表里详列的是其中和玩法、移动端、发布包直接相关的三份。2026-09-11。



发布包目录，web/ 展开一层，ZIP体积11.9 MB在第一行。「52个文件」这个数不用我说，选中的 build-report.json 在预览栏里自己写着："files": 52、"zipBytes": 11935880、"externalRuntimeDependencies": false。

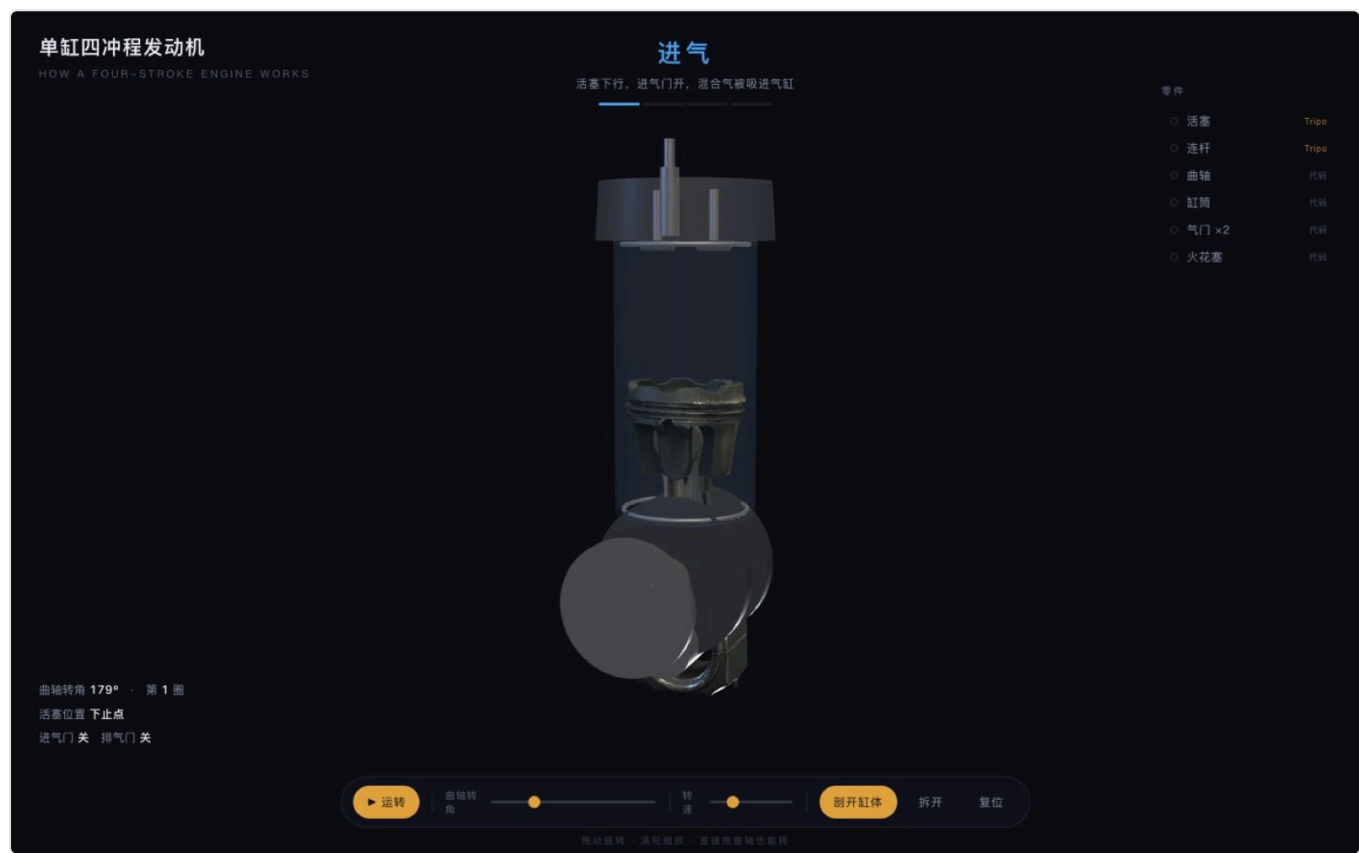
§20 机构类：一台能拆能装的发动机

Teaching Demos I: A Four-Stroke Engine You Can Take Apart

这节读完，你能自己判断一台机器里哪些零件该写代码、哪些该交给Tripo，并且知道怎么把「拆开看」升级成「装回去玩」。我用一个晚上做了一台单缸四冲程汽油机，一共只生成了两个零件。

先看结果

这台发动机能转、能剖开、能拆成六大件，还能让你按顺序装回去，装错了它会告诉你为什么装不进去。所有运动都是按曲柄滑块的公式实时算的，没有一帧是事先录好的动画。



发动机demo v1，2026-09-10晚。右上角的零件清单是这一节的主角：活塞和连杆标着Tripo，曲轴、缸筒、气门、火花塞标着「代码」。左下角的曲轴转角179°、下止点、气门开闭状态都是实时算出来的。



同一台机器，剖开缸体看内部。曲轴转角33°、活塞下行89%、进气门开。注意画面里的质感差别：活塞和连杆有环槽、有工字截面、有金属反光，下面那几个灰色圆柱是代码画的曲轴。

做它的起因很直接。前一天晚上我做完一版《星月夜》的立体拆解，自己的判断是「组件构成还是有些怪，效果不够理想，但是这种拆解的方式非常有趣」。既然有趣的是拆解本身，那就该找一个天生就该被拆开看的東西。发动机是最好的那个。

整台机器的运动只由一行公式决定，活塞销的位置从曲轴转角精确推出，连杆两端实时连上曲柄销和活塞销：

```
// R = 曲柄半径, L = 连杆长度, th = 曲轴转角
function pinPos(th){                                // 活塞销
    return new THREE.Vector3(0, R*Math.cos(th) + Math.sqrt(L*L -
R*R*Math.sin(th)*Math.sin(th)), 0);
}
```

四冲程的相位跟着走：曲轴转两圈（720°）是一个循环，进气门在0到180°开、排气门在540°到720°开，升程走一段正弦窗。这些都是查得到的机械常识，你不需要懂，Codex懂，你只要知道它必须由公式算、不能由动画摆，否则观众拖到某个角度就会发现活塞和气门打架。

那条分工判据

这台机器一共六种零件，只有两种是Tripo生成的。当时怎么分的，后来成了整本书里我最常回头用的一条判据。

零件	谁做的	为什么
活塞	Tripo生成	有环槽、有裙部、顶面有凹坑，形状复杂，手搓不划算

零件	谁做的	为什么
连杆	Tripo生成	工字截面加大小头圆环，同上
曲轴	代码	曲柄销的位置必须精确，偏一点整台机器就是错的
缸筒/缸盖	代码	回转体，代码更干净，还能做成半透明剖视
气门/火花塞	代码	同上，而且气门要按相位实时开合

判据：决定机构对错的尺寸件用代码，决定观感的形体件用Tripo。换个问法更好用：这个件做错了，懂行的人一眼能看出「机构不对」吗？能，就写代码。

这条判据后来在游乐园那一轮又验了一次，摩天轮的圈和辐条必须是正圆、过山车轨道必须闭合平滑，这些全归代码；吊舱、木马、茶杯这些只决定好不好看的，全归Tripo。这条判据是撞出来的，我一开始并没有它。

曲轴：我真的去生成过，然后放弃了

「曲轴该写代码」这句话，是花了50积分换来的。我先老老实实送了一张四缸曲轴的照片进Tripo。



准备好的单拐曲轴参考图，白底、四分之三视角、单个物体，按 § 02 那套配方裁的。



Tripo Studio里的曲轴结果，19228三角面/ 12580顶点，v3.1最高质量，8K贴图开着。好不好看？非常好看。能不能用？不能。

问题不在质量。**曲轴的功能全部藏在两个数字里**：曲柄销偏离主轴中心多少（决定行程），以及四个拐之间的相位差。这两个数字在生成出来的网格里测不出来，只能量个大概，而量个大概的曲轴装进机构，活塞的上下止点就全是错的。我可以写个脚本去网格里拟合销半径，那比直接用代码画一根曲轴贵十倍。

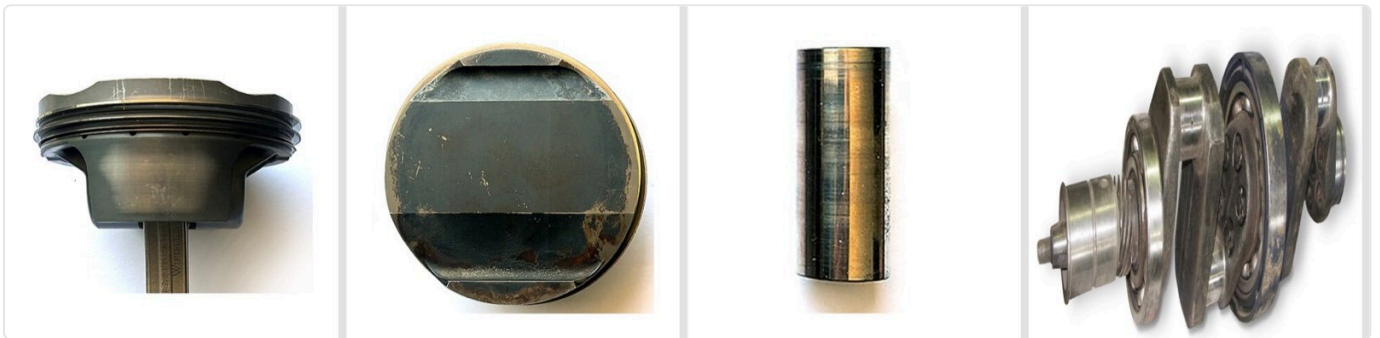
所以最后的成品里，那根曲轴是十几行代码画的几个圆柱，丑，但每一个尺寸都是我自己定的。**丑而正确，胜过好看而错误**，教学演示尤其如此，因为观众是冲着「机器到底怎么动」来的。

活塞和连杆：两张图，100积分

这两件走的是标准的图生3D。原始参考图取自维基，图页上标的是公有领域，我先在本地裁成单物体、白底、四分之三视角。（当时拿它生成模型和把成品随书再分发是两个标准：我没有逐张回核许可证版本与附加条件，所以这两件在随包素材里被剔掉了，口径见 \$ 00。）



原始参考照片：一整套活塞连杆组件，维基图页标注为公有领域。第一步先看清楚这个件由几部分组成，再谈怎么裁。



挑素材那一步。四张候选：活塞侧视、活塞顶视、活塞销、曲轴。挑图的标准是能不能看出这个件的比例和特征，清晰度排第二。



实际送进Tripo的活塞图。这张图埋着后面那个坑：为了让活塞顶部占满画面，我把裙部下半截裁掉了，Tripo学到的比例因此是扁的。



连杆的输入图。大小头两个圆环和中间的工字截面都完整入画，这一张就没出比例问题。

两次生成各50积分，一共100。按Pro套餐¥140 / 3000积分换算，这台发动机的模型成本是**4.7元**。

耗时口径这一节和后面两节通用：本书引用平台侧的秒数一律是**算子耗时**（算子耗时口径，不含上传与排队；端到端要往上加，两个口径的实测差见 § 01，词条见附录D）。你自己坐在界面前等的是另一个数，中间夹着上传、排队、出预览、下载。

把生成件接进机构的三条手艺

Tripo给你的的是一个模型，机构要的是一个带着安装接口的零件。中间这一步没人替你做，好在只有三件事：

1 销轴方向：取包围盒最短的那根轴

生成件的朝向是随机的，而连杆必须绕着销子转。做法是量出模型的包围盒（这类词不认识就翻附录D，那里按你会撞上它们的顺序讲了一遍人话），最短的那根轴就是销轴方向，把它转到Z轴。连杆实测是 $0.293 \times 1.0 \times 0.39$ ，最短轴正是销轴，一次就对。

2 孔心位置：从顶点分布里估

连杆两端的孔心在哪，模型自己不会告诉你。做法是取上下各22%的顶点，分别求它们到质心的横向最大值，当作端部外径，孔心大约就在「端部极值减去外径」的位置。估出来的销距和真实值吻合，缩放之后整个机构闭合，活塞不会脱出缸筒。

3 活塞纵向拉伸1.75倍

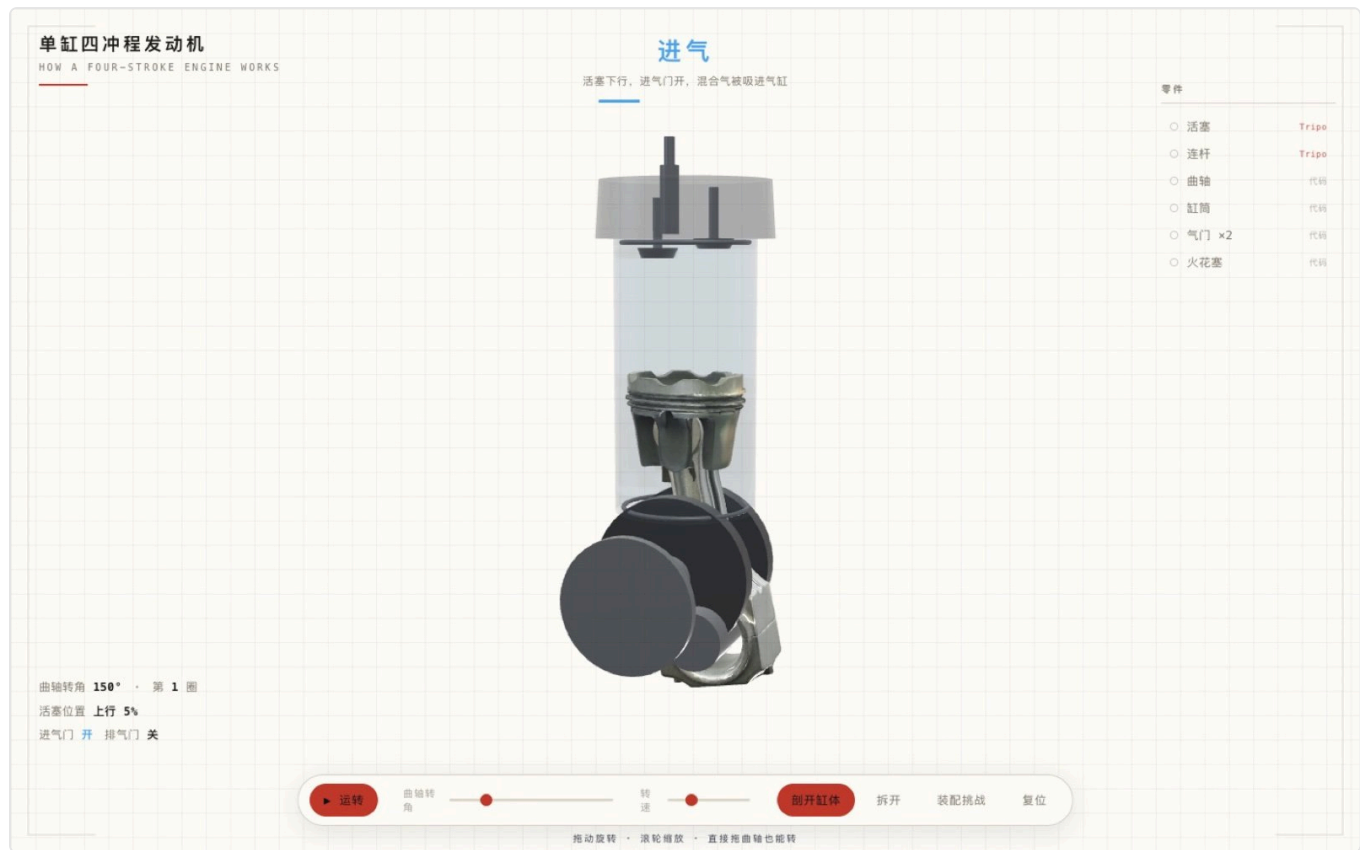
上面那张裁过的输入图的后果：生成出来的活塞是扁的，直接摆进去像一枚硬币。沿Y方向拉伸1.75倍才回到真实活塞的比例。**这是整台机器里唯一需要人工校正的地方，而且只在一个方向上。**

核心建议

第三条其实是第 § 02节那条规矩的反面教材：喂给Tripo的图，比例不能裁。它学不到你裁掉的那部分，只会照着你给的框去理解这个物体有多高。

白底那一课

v1是深色界面。做v2时我想换成纸感白底，理由在X上那一轮案例分析里：同期高赞的3D demo几乎全是深色赛博风，已经有人在公开抱怨审美疲劳，而拿到10060赞的@tomkrcha那条《蒸汽图鉴》(<https://x.com/tomkrcha/status/2096082580554777041>，2026-09-05)是白底、衬线排版、编辑设计级的。一片深色里，白底本身就是差异化。



v2的纸感白底版：米白#fdfcf7加细网格加四角裁切线，朱红换掉了原来的琥珀色。底部控制条多了「装配挑战」。

换白底那一下，画面直接黑了。

注意

白底+ PBR金属+没有环境贴图=渲染成死黑。PBR是基于物理的渲染（附录D），材质带金属度和粗糙度两个参数；金属度高，漫反射就被抑制，画面里的亮度几乎全靠反射环境得来。没有环境贴图，它反射了个寂寞。深色背景下这个问题被藏住了，一换白底就原形毕露。

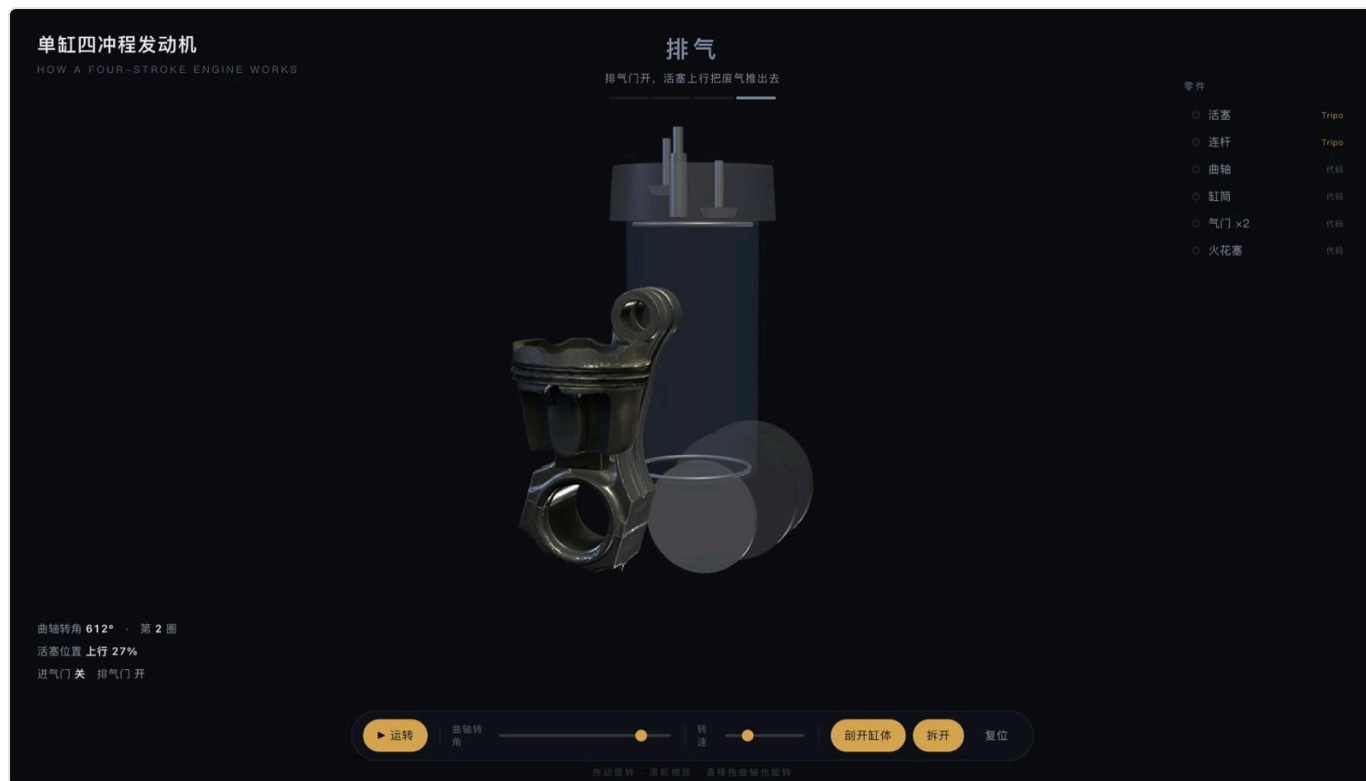
解法不需要任何外部HDR文件：用canvas画一张从白到深灰的渐变，当成全景环境贴图喂给PMREMGGenerator，再赋给 scene.environment。完整代码和三个配套细节（渐变必须留一块暗部当黑旗、白底别开ACESFilmic、金属度压下来）在 § 12，附录B第一条也收了这段代码。这台机器上唯一要补的是那个数值：Tripo的PBR材质默认金属度偏高，实测压到0.85以下、同时补高环境光强度才对得上。

注意

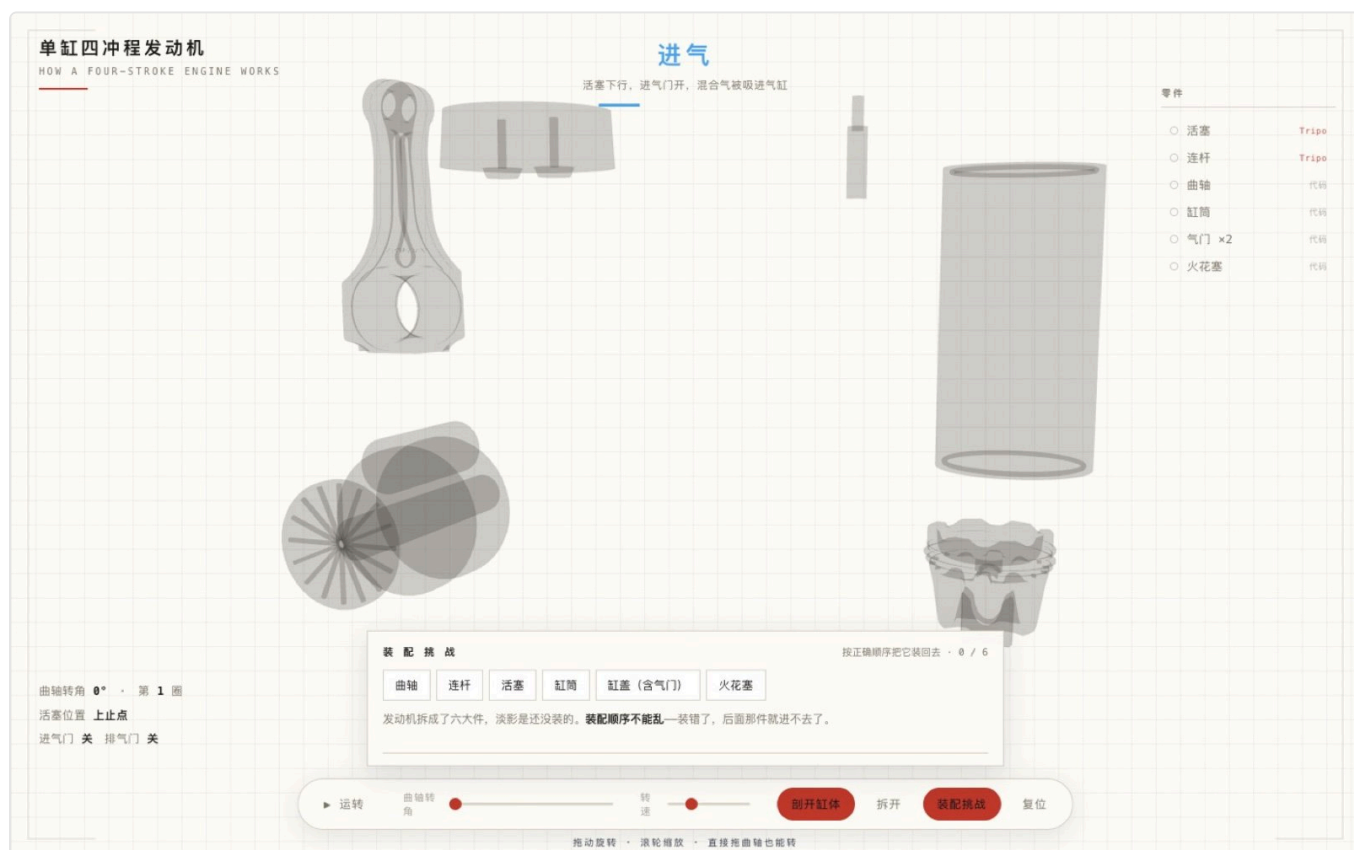
这上面我还栽过一跤，和材质无关。调了五轮视觉，前四轮我都在判断「还是发白」，然后继续调环境贴图、调曝光、调金属度，真正的原因是页面上那层不透明白色的加载遮罩没淡出。截图自校验之前，先确认加载遮罩已经消失。判据和修法在 § 12。

从「拆开看」到「装回去」

v1做完我就写下了一条自评：拆解是「看」，装配才是「玩」。这台机器当时只能拆开、合上、转一转，缺的是那条「装错了会卡」的交互。v2把它补上了。



拆开状态。这是「看」：六大件飞散开，你能转着看每一件长什么样，但你不需要理解任何东西。



装配挑战模式，进度0/6。六件以淡影摆在画面四周的待装位上，点对一件，它就从待装位飞回原位并变成实体。

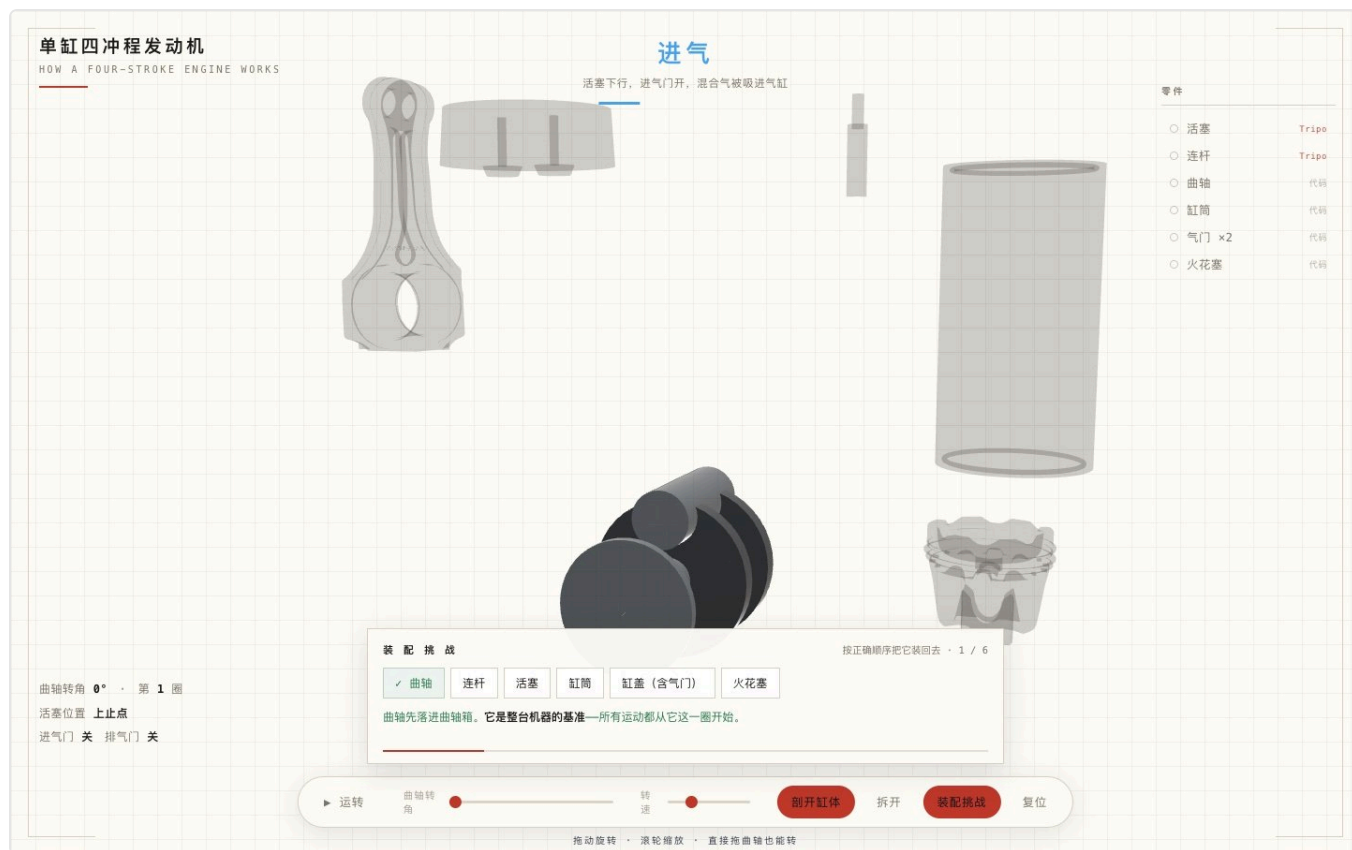
正确顺序是曲轴、连杆、活塞、缸筒、缸盖（含气门）、火花塞。点错了不扣分，弹出来的是一条真实的机械约束：

- 先点连杆：「曲轴还没进去，连杆大头得抱在曲柄销上，销子都还没有，它抱谁？」

- 先点活塞：「活塞得靠活塞销挂在连杆小头上，不然它在缸筒里没东西带动，纯粹是块铁。」
- 先点缸筒：「缸筒一罩下来，活塞就没法从上面压进去了，真实装配里这一步装反了就得全拆。」
- 先点缸盖：「缸盖是压在缸筒上端面的，下面悬空压什么？」
- 先点火花塞：「火花塞是拧在缸盖上的螺纹孔里的。」



装配模式里先点连杆：提示条实读「装不上去 —— 曲轴还没进去。连杆大头得抱在曲柄销上——销子都没有，它抱谁？」，同时连杆那个按钮进抖动状态。点错不扣分，弹出来的是一条真实的机械约束。2026-09-11实拍。



接着点曲轴，飞入动画结束后曲轴由淡影转为实体，进度条走到1/6。提示条不夸你，直接讲下一层：「曲轴先落进曲轴箱。它是整台机器的基准——所有运动都从它这一圈开始。」2026-09-11实拍。

装对了它也不夸你，接着讲这一件为什么长成这样：连杆的工字截面是干嘛的、燃烧室到底指哪块空间。六件齐了自动开始运转，收尾一句：这个顺序是几何定死的，每一件都挡着后一件的路。

为什么装配比拆解值钱：拆解只能告诉你有什么，装配会逼着你回答为什么。要装对，你就得知道每一件挡着谁的路。同期那条10060赞的《蒸汽图鉴》能拆开合上看参数，装配这一步是我这边多做的。

装配模式实现时踩的两个坑

第一个：偏移不能在每帧的位置计算里累加。曲轴、连杆、活塞每帧都被运动学重新赋值，给它们加偏移没事，下一帧会重置；但缸筒、缸盖、火花塞是初始化时设一次的静态件，每帧 += 会让它们一路飞出屏幕。气门更麻烦，它的y每帧被升程动画覆盖，加了也白加。

解法是让偏移只在渲染那一刻生效：

```
asmApply();           // 记录原位，加偏移
renderer.render(scene, camera);
asmRestore();         // 渲染完立刻还原
```

第二个：凡是 scene.add 进去的东西都要有归属。缸筒内壁那两道细环是独立的mesh，一开始没归进「缸筒」这一件，于是装配模式下缸筒明明还没装，画面上却飘着两道环。

v2.1：空画布和隐形的淡影

装配模式上线后第二天复核，一进去画布全空，第一眼像页面坏了。改成六件以淡影留在画面里的待装位，进装配模式时相机先归位（淡影的摆位是按默认机位排的）。又踩两个坑：

- 淡影必须用不受光材质。这个场景四盏灯加环境贴图，亮度很足，MeshStandardMaterial 的中灰会被打成近白，在白底上等于隐形。
- 火花塞里的「火花」不能和淡影共用一份材质。它每帧被写 opacity，共用之后整份淡影材质被抹成全透明。症状是「材质挂上了、visible 也是true、就是看不见」，靠页尾挂的 window.__eng 探针才查出来。

这一台的账

项	数
Tripo生成	2次（活塞、连杆），各50积分
小计	100积分 ≈ ¥4.7
另计	曲轴生成1次，留在可行性测试目录，没进demo
代码件	曲轴、缸筒、缸盖、气门 ×2、火花塞，0积分

一台能转、能拆、能装、能讲出四个冲程的教学演示，模型这一块的全部花费是四块七。贵的从来不是生成，是你决定生成什么。

核心建议

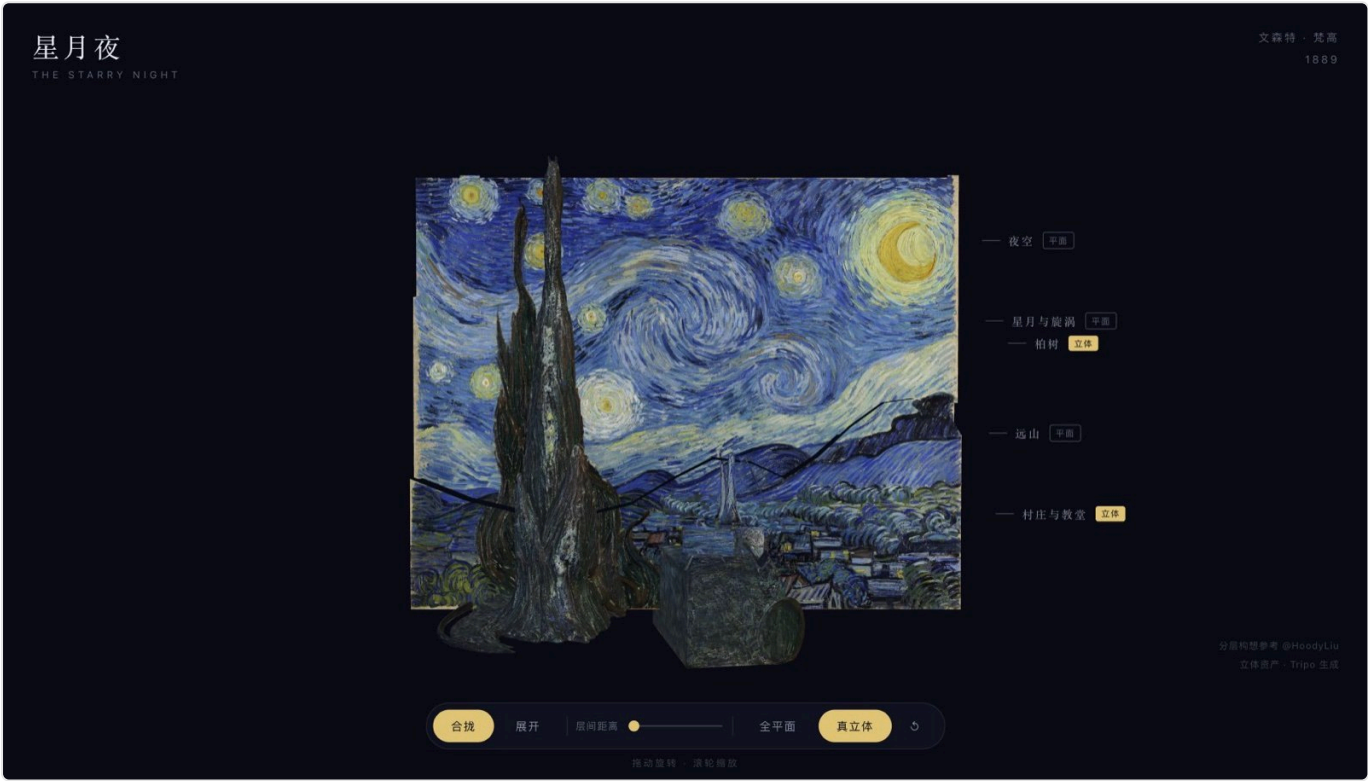
迁移到你自己的题目上，就问这一句：这台机器里，哪几个件「做错了懂行的人一眼能看出来」？那几件写代码，剩下的全部交给Tripo。齿轮箱、缝纫机、锁芯、活塞泵，全都适用。

§21 艺术类：把一幅画拆成五层

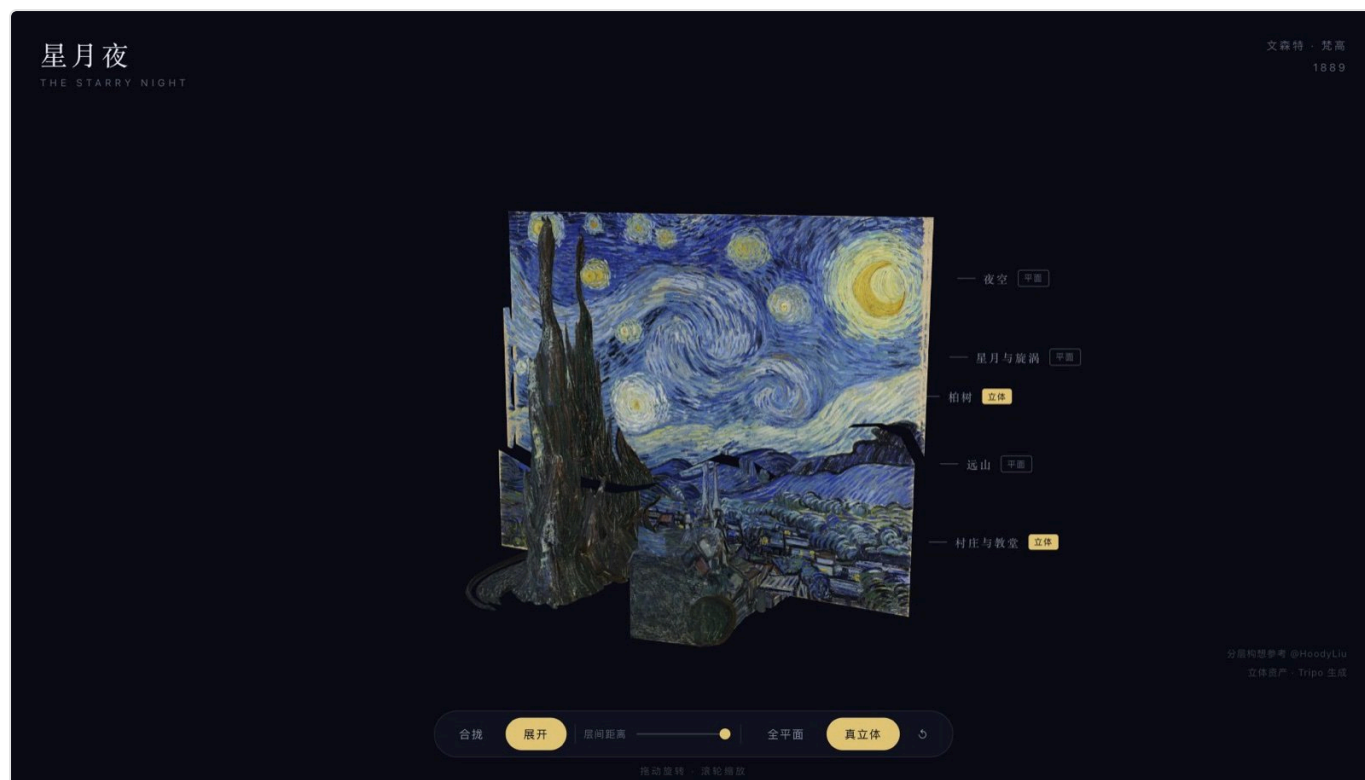
Teaching Demos II: Unfolding a Painting into Five Layers

这节读完，你能判断一幅画里哪一层该保持平面、哪一层该做成真的立体，并且知道Tripo在艺术素材上能保住什么、会丢掉什么。用的是《星月夜》，19分钟做出第一版，全部花费200积分。

先看结果



合拢状态。正面看就是那幅画，五层严丝合缝地叠在一起，你看不出任何异常。



展开状态。五层沿Z轴分开，镜头同时侧过身。右侧标签标着每层的处理方式：夜空、星月与漩涡、远山是平面，柏树和村庄与教堂是立体。右下角标注了分层构想的来源与立体资产的出处。

起因是2026-09-10我在X上刷到@HoodyLiu的一条 (<https://x.com/HoodyLiu/status/2097536045881704688>)，用Three.js把《星月夜》做成了可以拆开的「3D油画」。我用1440p原片逐帧看了一遍，它的真身是**五层带alpha的平面抠图**，在Z轴上等距错开，加一个展开合拢按钮和一个层间距离滑块。他自己也说了灵感来自乐高版。

这个形式的软肋很清楚：柏树和教堂是一张竖着的纸。背景层平着没问题，它们本来就远；主体层一平，整个立体感就塌了。

分工是这个作品形式自己长出来的：「哪些层该平、哪些层该立」是判断，交给编程Agent去规划；「把那一层真的做成立体」是形体，交给Tripo。

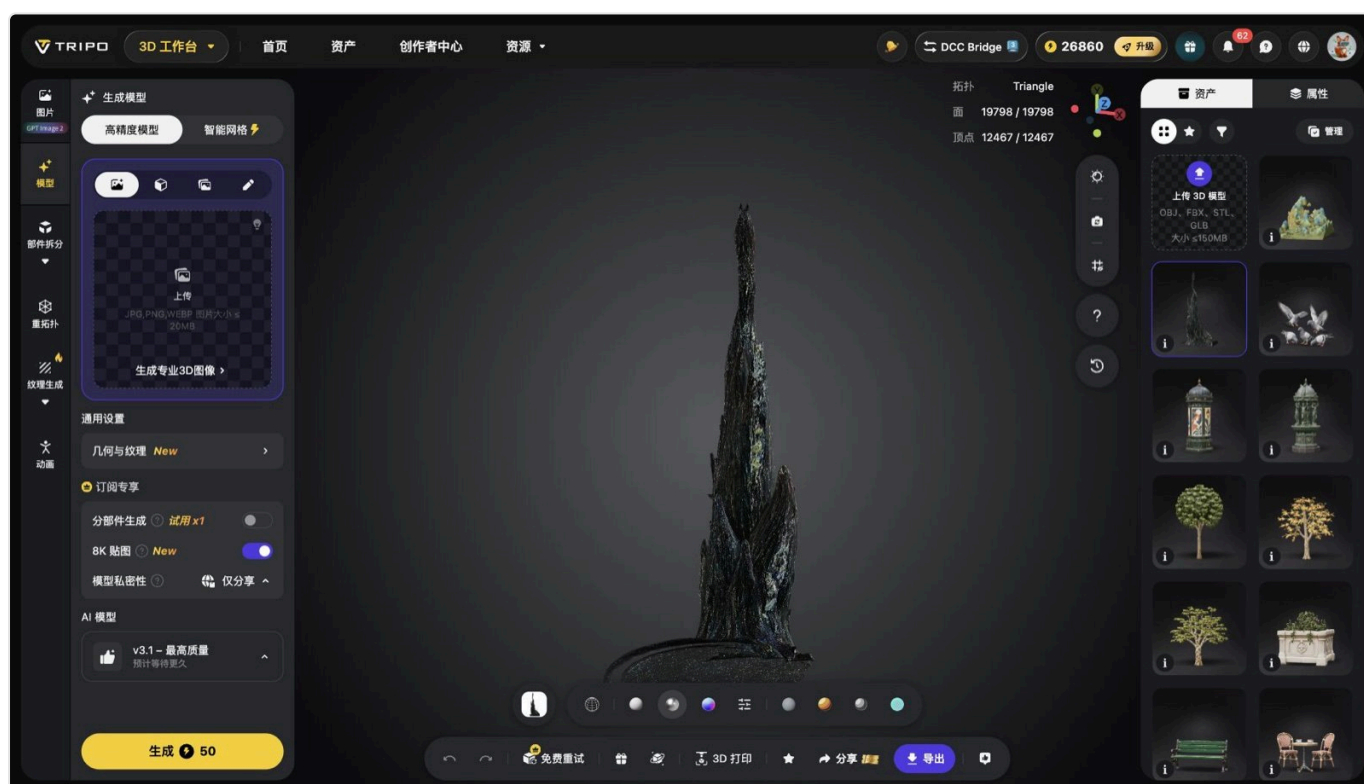
先花100积分回答一个问题

动手做demo之前，有一个问题不回答就白做：**Tripo能不能把画里的一层做成真3D，同时保住画家的笔触和颜色？**能，这条路成立；不能，整条路作废。

所以我先做了两次可行性测试，一次拿梵高的柏树，一次拿《千里江山图》的一段山，两个画种、两种笔法，看看边界在哪。两次都是v3.1最高质量加8K贴图，各约四分钟。**这个四分钟是我坐在界面前等的墙钟，不是算子耗时**，里面含着上传、排队和下载。两个口径差多少，\$20那条已经量过了：现场实测生成端到端约31秒，其中算子16秒。本书引用平台侧的秒数一律按算子耗时口径。



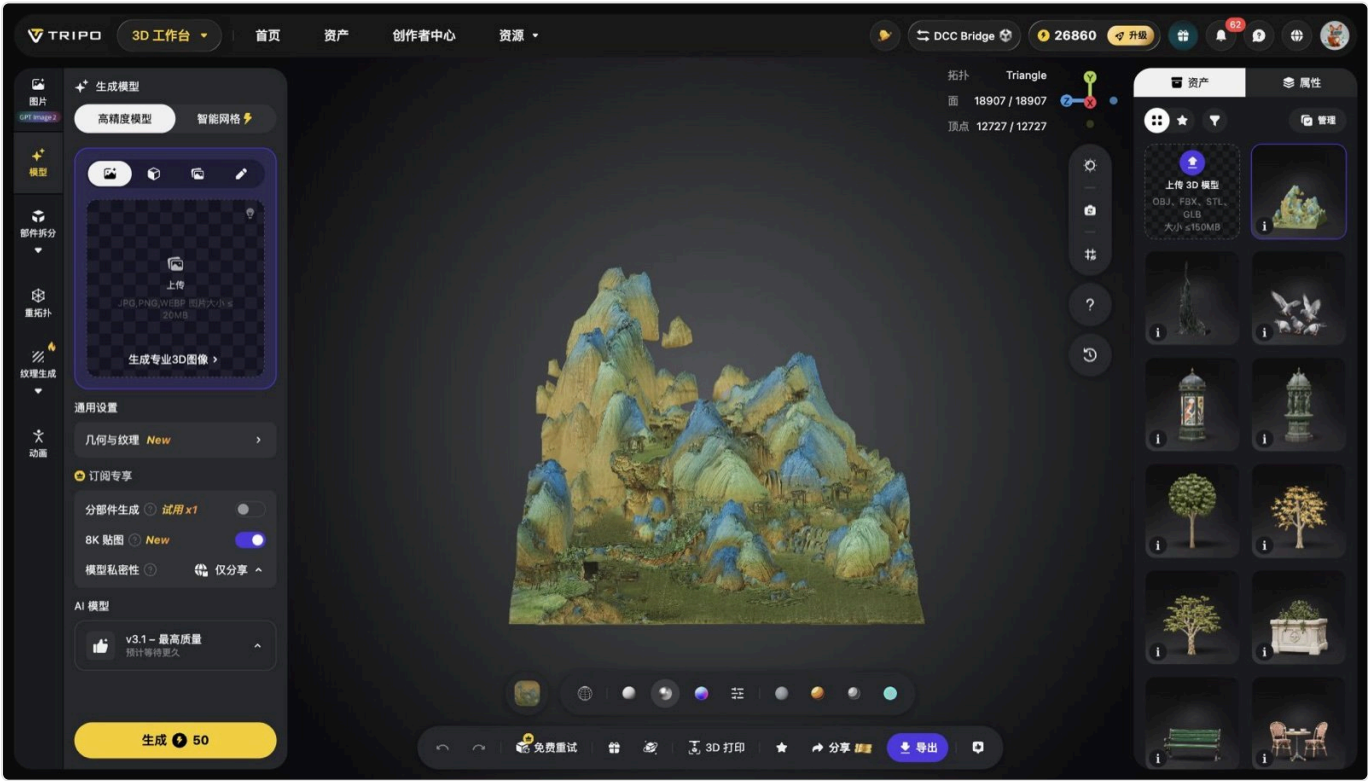
输入之一：从3840 px原图里手工抠出柏树，去掉背景，缩成白底1024²。抠图这一步没有偷懒的余地，背景留一点，Tripo就会把它当成物体的一部分长进去。



柏树的生成结果，19798三角面/ 12467顶点。侧视确认它有真体积：地面底座是张开的，尖塔有厚度。但形体已经不是柏树了。



输入之二：《千里江山图》里截的750×700一段山，同样缩成1024²。



山段的生成结果，18907三角面/ 12727顶点。山体有真实起伏，石青峰、石绿坡、赭色绢地全部原样保留，底面是一个裁切平面。

	星月夜 · 柏树	千里江山图 · 山段
面数/顶点	19798 / 12467	18907 / 12727

	星月夜·柏树	千里江山图·山段
体积	真体积，底座张开、尖塔有厚度	真浮雕，山体有起伏
颜色与笔触	保住，油画笔触清晰可见	完全保住，矿物色原样
Tripo自动命名	rocky spire 3d model	traditional landscape painting 3d model
问题	形体语义漂移：柏树被理解成了岩石尖塔	没有「底」，出来是一块矩形浮雕板

三个结论，一个都不是坏消息

风格保真度不是问题。Tripo用输入图烘焙纹理，笔触和矿物色原样保留。这是整条路能不能成立的分水岭，答案是能。

形体语义会漂移。柏树出来叫 rocky spire，火焰般的枝叶变成了一整块尖塔，根因在 S 02。顺带说一句，rocky spire 这个名字本身就是一条免费的验收信号——Tripo给每个模型自动起的名字就是它对形体的理解，名字不对，模型八成也不对，不用渲染就知道。

地景比物件更需要一个「底」。山段出来是一块矩形浮雕板，这其实是对的方向：它变成了一件青绿山水雕件，玉山、木雕山水，都是中国人本来就熟悉的形态。把国画立体化，其实就是把它还原成它最容易被立体化的那个形态。

核心建议

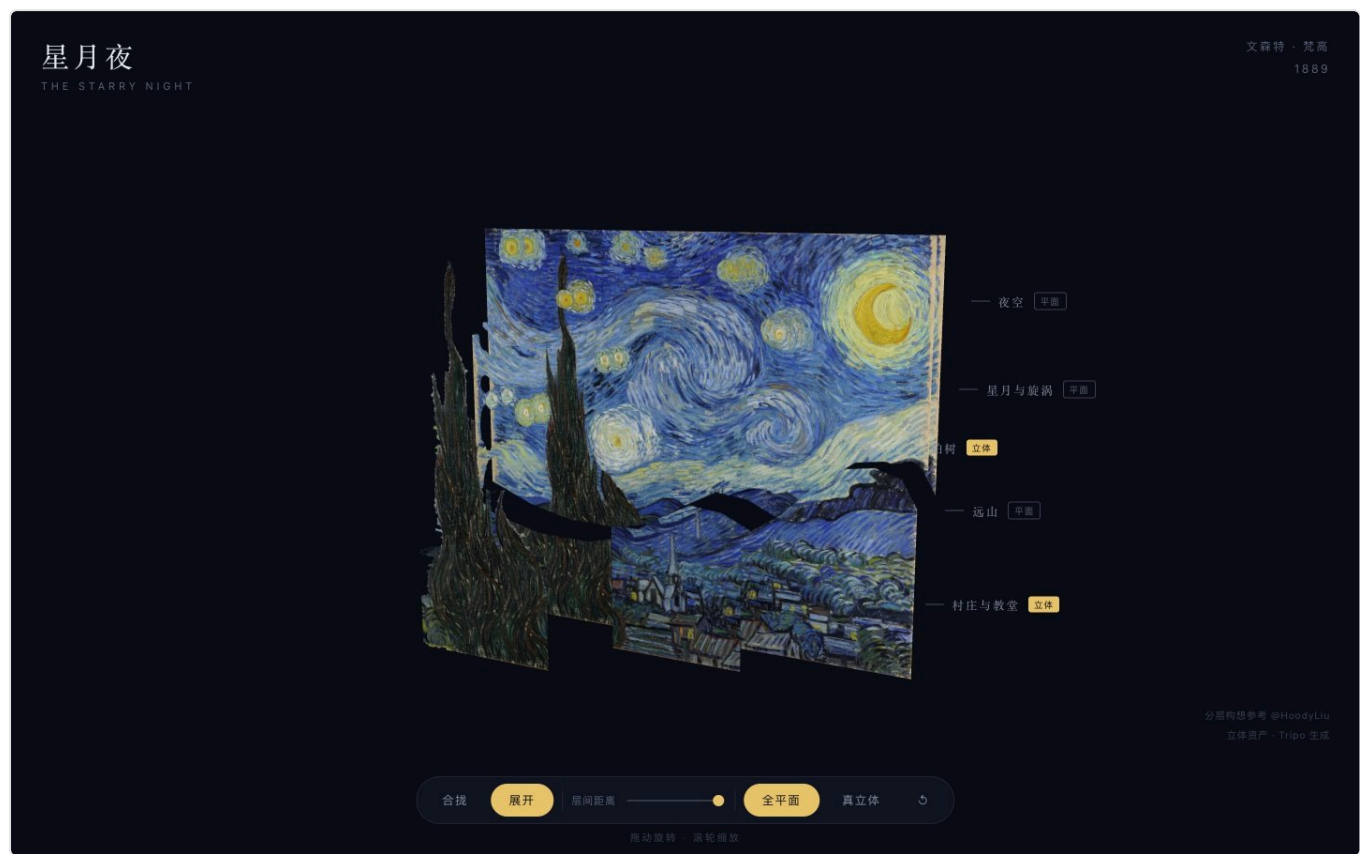
两个结果都能用，而且各自暴露了不同的边界。做实测的时候别急着把不完美的那次删掉，边界本身就是内容。

哪层该平，哪层该立

可行性过了，回到那幅画。五层同名同构，但其中两层的判断和原帖不一样：

层	处理	为什么
夜空	平面	最远的一层，它只需要颜色
星月与旋涡	平面	它们是光，本来就没有实体
远山	平面	远到空气已经把细节吃掉了
村庄与教堂	Tripo真3D	有正立面、有前后、有轮廓，一张纸装不下
柏树	Tripo真3D	离观众最近，而且它挡着后面的东西

判据其实只有两条：这一层有没有正面和侧面之分，以及它挡不挡别的东西。远山既没有侧面、也不挡谁，做成立体纯属浪费。



「全平面」模式下的展开态。和 p21-open.jpg 那张真立体对照着看：同样的展开动作，这里的柏树和村庄是两片立着的平面切片，那里是有体积、有正侧面的Tripo模型。这个按钮存在的意义就是让观众自己做这个对照，不需要旁白解释。2026-09-11实拍。

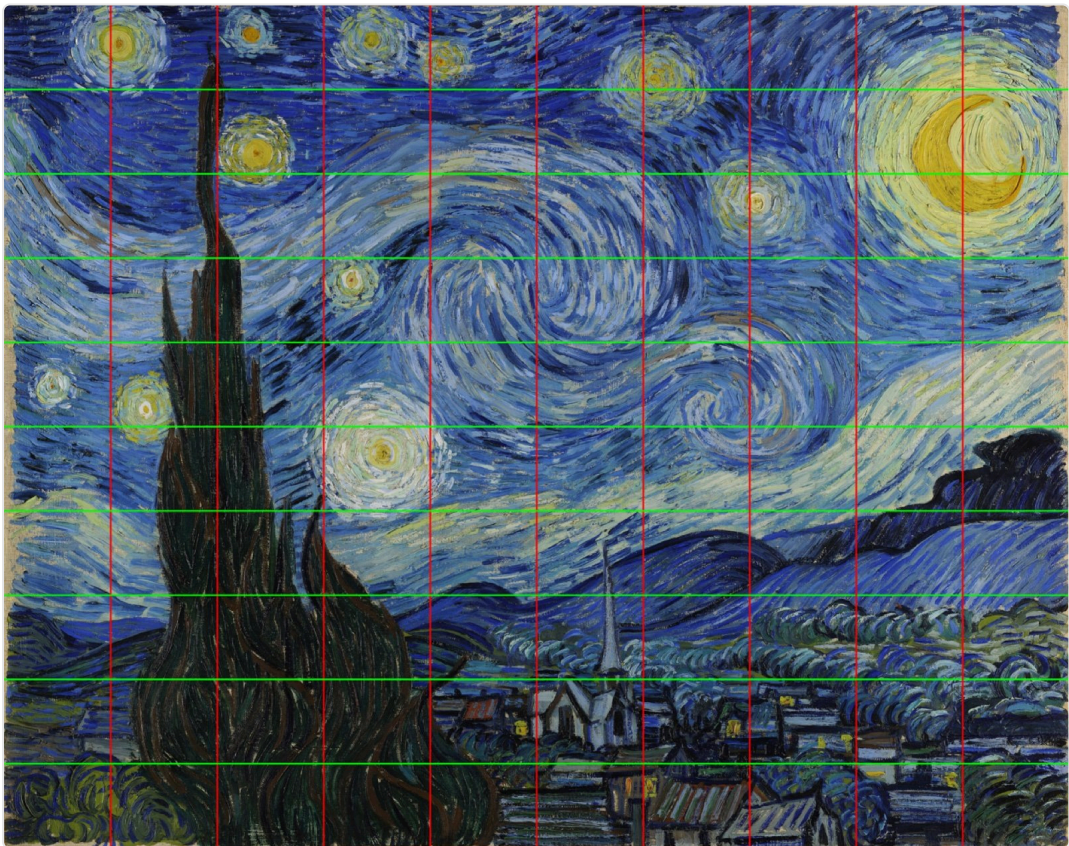
demo里因此有两个按钮，展开合拢之外，还有一个「全平面/真立体」，一按就把那两条立体层换回画里的平面抠图。差异摆在眼前，观众自己判断，不需要旁白解释。

怎么把一幅画切成图层

这一步没有AI参与，纯手工，但它决定了后面所有事。



起点：3840 px的《星月夜》原图（1889年，公有领域）。所有图层都是从这一张里裁出来的，越早拿到高分辨率原图，后面越省事。



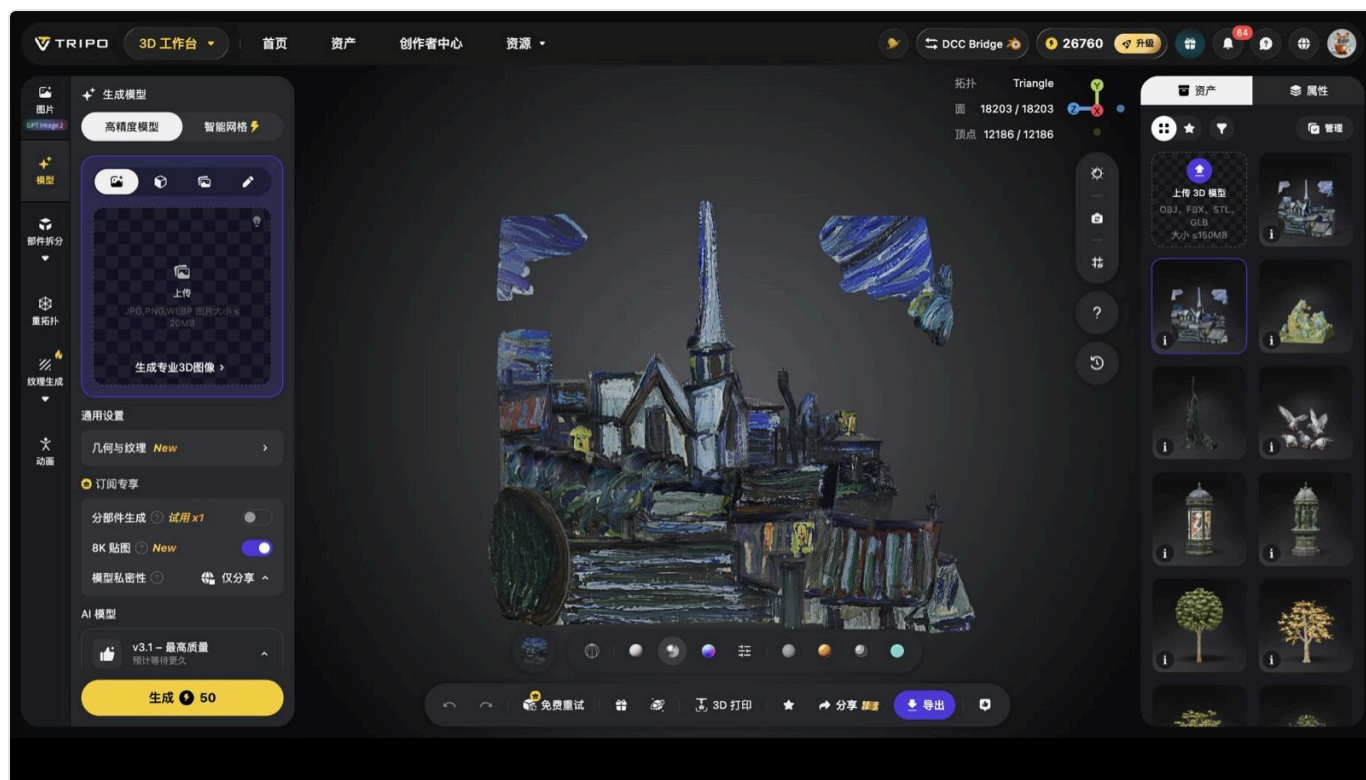
给原画叠一层网格，用来定裁切坐标。分层之前得先能准确说出「村庄在图的哪个矩形里」，靠肉眼描述来回改三轮都对不齐。



裁出来的村庄条带。这一条里有教堂尖顶、有屋顶层次、有前后遮挡，是全画最该立体的一层。



实际送进Tripo的输入：把条带羽化成一个椭圆，压在白底上。羽化是为了让它别把这一条当成一块四方的板去理解（这是我的做法，没有做过对照实验）。



生成出来的村庄与教堂。房屋有前后层次，教堂尖顶立起来了。它偏暗，因为源图本来就是夜景。

切出来的资产分两套：三张平面层是带alpha通道的全分辨率PNG（夜空、星月与旋涡、远山），立体层除了GLB，还要另存一份平面版的抠图。后面那份就是「全平面/真立体」按钮要切的东西，一开始没做，后来发现没有它就没法把差异摆给观众看，只能靠嘴说。

整个页面是一个不依赖CDN的静态目录，three.js本地内置，两个GLB各约9到10 MB。但它必须走http打开，`python3 -m http.server` 起一个就行，双击用 `file://` 打开会被CORS挡住，GLB根本加载不了（纯静态、无外部资源的HTML不受此限，这也是很多人第一次做3D网页时最先撞的墙）。

四条只有做过才知道的

1 底层不要挖洞

直觉上，立体层既然立在前面，底层对应的位置就该挖掉，不然会重影。我一开始就是这么做的，结果展开时出现黑方块，洞的边缘和3D模型的轮廓永远对不齐，非常难看。改成底层保留完整画面、3D物体立在它前面之后，合拢时完全重合，展开时画里的那一层留在后面像一个影子，反而更好看。

2 展开必须带镜头位移

正视角下五层分开你根本看不出来，只是画面糊了一点。展开动画要让镜头同时侧过身，分层才成立。这一条是必须的。

3 金属度按到0

Tripo导出的PBR材质（基于物理的渲染，材质带金属度和粗糙度，附录D）带金属度，Three.js里没有环境贴图就会渲成一片死黑。这里我用的是加载后直接 `metalness = 0`、`roughness ≈ 0.92`。不加这一条，村庄模型就是一团黑。

4 按「覆盖」缩放，别按「装下」

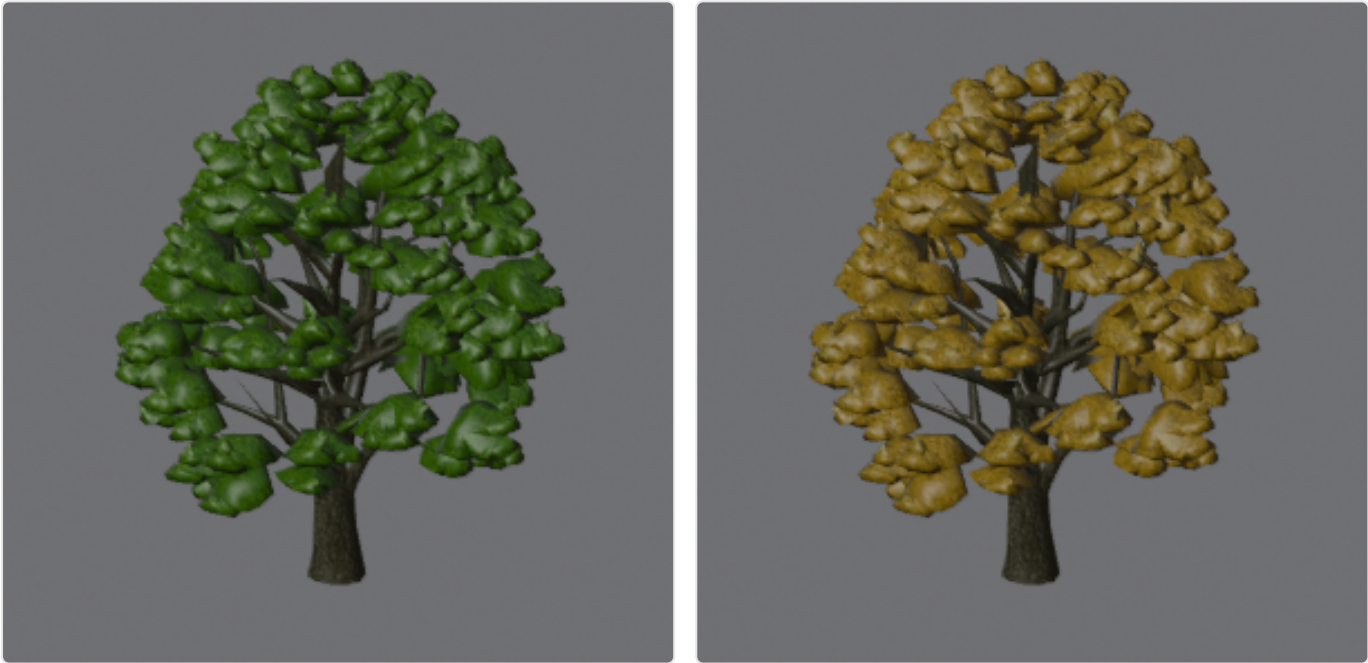
把模型缩放到对应位置时，取三个方向缩放比的 `Math.max` 而不是 `Math.min`。用min它会完整地待在框里，但盖不住画上原来那块，边缘漏出来。

秋叶那一课：改一张贴图，等于多一件资产

这一条发生在巴黎那个城市demo里，但它讲的是「怎么调一张画出来的颜色」，方法论上属于这一节，所以放这儿。

巴黎那条街上要两种行道树，绿叶的和秋黄的。Tripo那边只生成了一棵七叶树。按常规做法，另一棵是再送一张秋天的设计图进去，再付一次生成加一次贴图的钱。但这两棵树的几何本来就该完全一样，差别只在叶子的颜色。所以我换了个做法：不重新生成，直接改那张基色贴图。

这件事在文件层面很干净。GLB里顶点数据和贴图是两段独立的字节，重建贴图那一段、其余原样搬运，几何就不可能变。这不是「事后核验发现没变」，是构造上就动不了。三角面11,825，改色前后逐位相同，花费0积分。



左： `tree-01-maronnier-green`。右： `tree-02-maronnier-autumn`。同一个网格，11,825三角面逐位相同，右边这棵只是把基色贴图里叶子那一段颜色挪了位置，没有再走一次Tripo。本地脚本批量渲图，2026-09-12。

难的是颜色本身。第一版我按直觉调：把叶子那段的色相推到28°到51°那一带，同时把饱和度推高18%，想着秋天嘛，颜色总该更浓一点。出来是一片铁锈褐，像枯死的，不像秋天。

正解和直觉正好反着来：

	第一版（错的）	正解
目标色相	28°-51°	45°-54°

	第一版（错的）	正解
饱和度	推高18%	压低8%
明度	不动	拉高38%
出来是	铁锈褐，像枯死的	金黄

秋叶的关键不是更饱和，是更亮更黄。把饱和度往上推只会让它掉进「烂掉的叶子」那一档，因为深褐本来就是高饱和低明度的黄。这个直觉我一直是对的，是这次被一棵树纠正过来的。

还有两个细节，不处理这条路就走不通。

- 1

色相窗口要收窄，不然把树干也染了

第一版我从71°开始算「这是绿色」，结果树皮那种偏黄绿的棕也落在窗口里，跟着一起被推成了黄色。往上收到85°起才干净。判据是去看你要动的那个颜色和你不想动的那个颜色在色相环上隔多远，隔得近就得把窗口收到它们中间，宁可漏改一点边缘。
- 2

只能改基色那一张，别碰法线和ORM

第一版我把这个模型挂着的贴图全部改了一遍。法线图和ORM图（附录D）里的RGB根本不是颜色，是表面朝向和材质参数，拿去挪色相等于把材质写坏。正确做法是先从材质里读出哪一张挂在基色槽位上，只改那一张；读不出来就直接报错退出，不做盲改。这件事在P2.0出的模型上不会咬你（它本来就只有一张基色），但v3.1那批三张齐全的一改就毁。

核心建议

这个做法的适用面比它看起来大。凡是「同一个形体、不同颜色」的需求，都不该是一次新的生成：同款不同色的车、四季的同一棵树、白天和夜里的同一栋楼。生成的钱花在形体上，颜色留给贴图。

选画：不是每幅都能这么拆

做完这一版，「下一幅画选什么」就有判据了：构图本身是不是深度分层的。

推荐

《星月夜》是。近景柏树、中景村庄、远景山、天空，四层叠得清清楚楚，天生适合「拆开」。

不推荐

《清明上河图》不是。它是横向长卷，深度只有薄薄一层，硬拆成五层也拆不出东西。这类得换个动作：走进

还有一条边界要提前说：**雕塑和建筑本来就是3D的**，对它们做「3D化」这个动作根本不成立，得换动作，换素材没用。

这一幅的账，以及那50积分的教训

这张账我只能给到总账这一级，原因写在表后面。

项	读数
开工前余额	26910
收工后余额	26710
本轮消耗（含两次可行性测试）	200积分 ≈ ¥9.3
这200积分跑了几次生成	5次
逐笔单价	未记录

上面那两个余额是当时从界面上抄下来的实读值，5次生成也是实数。但逐笔是哪一次花了多少，我当时没有一笔一笔记。所以别去做 $200 \div 5$ 这道除法：它得出的40只是个平均数，不等于任何一档的单价，同一轮里不同贴图档位的价格本来就不一样。这本书里凡是出现「这一件多少积分」的地方，背后都有一条对得上的余额流水；这一幅没有，那就只报总账。有总账没逐笔就直说，比硬凑一张看起来很完整的表可信。

注意

点完「生成」，积分没立刻变、页面没跳转，不等于失败。我判成失败又点了一次，结果是两条任务同时在跑，白花50积分。等几秒再去复查任务列表，不要立刻重试。

可行性过关之后，第一版能点的成品只花了19分钟。这个数字我自己也愣了一下。制作是快的，慢的是前面那一下判断：哪层该立。而判断不花积分。

已知不足

这一版摆在这儿的问题，我没打算藏：3D柏树比画里那棵略大略高，合拢时画里的会露出一一点边；村庄模型偏暗、辨识度不如柏树；星月层的Z间隔太小，展开后几乎看不出它是独立一层；每层点开只有一句我自己的判断，还没接梵高书信原文（要接就必须去vangoghletters.org逐句核实，不能凭印象写）；移动端没测。

核心建议

下一步最值得试的一件事：把原画的对应区域当贴图回贴到生成出来的模型上，看能不能修正柏树那个形体漂移。风格保真已经成立，那么形体漂移也许可以用贴图兜回来一部分。这条我还没验证过。

§22 世界类：一颗你走得完的星球

Teaching Demos III: Building a Small World You Can Live In

这节读完，你能用一条流水线把十几个资产成批做出来并接进一个能走进去的世界，也会知道这类项目真正的瓶颈在哪（提示：不是三角形）。我用《小王子》做了三个版本的星球，16个资产全部一次命中，840积分。

注意

先说版权。圣埃克苏佩里1944年去世，按中国著作权法（作者死后50年）原著法语文本早已进入公有领域，美国那边要到2039年才进入公有领域（1943年首版，按95年保护期算；2019年进入公有领域的是1923年的出版物，别把这两个数记混了，我一开始就记混了），法国本土的状态也有争议，战时延长条款可能把保护期推到2033年前后。我的demo里所有法语引文都逐字核对过公有领域底本，中文是我自己的直译，不是任何一本已出版译本的句子（中译本的著作权独立，要用正式译本必须单独谈授权）。本书讲的是方法与流水线，这一节涉及的模型与文本资源不随书分发。你换成自己的题材做，这一步的功课一样要做。

先看结果：同一本书，三种星球观



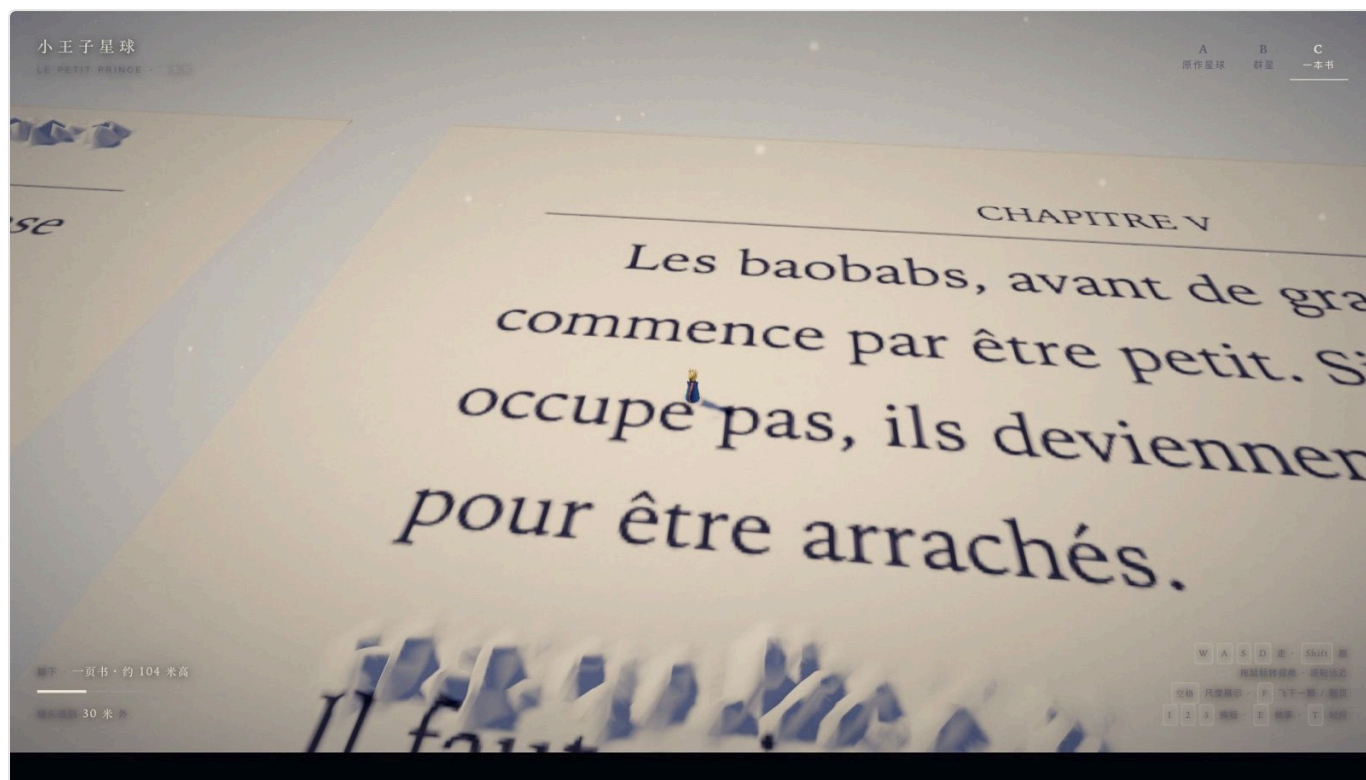
入场页。三种星球观做成三张卡片，底部一行「Tripo资产到位16/16」是引擎在建世界之前先把资产拉齐的状态提示。



A观，原作星球。就是B612那一颗，直径28米，绕着走一圈几十秒，玫瑰、火山、猴面包树苗、井、箱子都在它该在的地方。



B观，群星。七颗小星球挂在同一个虚空里，每颗住着一个大人。左下角HUD：脚下星球直径22米，镜头退到150米外，七颗一次全看见。



C观，一本书。字是地形，每个字母是一道真实的山脊，人走在字里行间。左下角写着「脚下·一页书·约104米高」。

为什么做三个。星球观不定下来，后面全是白做，而它又是看图才定得了的判断。与其写三段文字让人想象，不如三个都做出来，同一个页面按 1 2 3 切换。让选择变成三个能走进去的東西，而不是三段形容词。

为什么是「世界」而不是「地方」：游乐园是一个地方，你能做的是逛；星球是一个世界，你做的是探索。而这本书天然解决了「为什么这些东西要放在一起」，原作结构本身就是一份园区设计图。

流水线：Codex出设计图，Tripo出3D

这一轮定的规矩是，每一个摆进世界的模型都先由Codex生成一张精细设计图，再送Tripo生成3D。这条流水线本身就是要验的东西。



Codex出的设计图（部分）。统一的水彩加钢笔线风格、纯白背景、单个物体、完整入画。这套图既是Tripo的输入，也是整个世界的风格基准。



第二批：七位大人各自的道具。王座、镜子、酒瓶和桌、商人的书桌、地理学家的书桌、箱子、椅子。同一套风格锚，出来的模型才能摆进同一个世界。

端到端跑下来，15个静态资产加1个绑骨角色，16个全部正确。验收方式是我最推荐的那一条：**读Tripo给模型自动起的名字**。它给的slug就是它对形体的理解，名字里出现的东西，模型上都有。

资产	Tripo认成了什么	目标尺寸
玫瑰	red rose with green stem and leaves, botanical illustration	0.85
狐狸	fox illustration with orange fur, white chest and black legs	0.95
水井	stone well with wooden frame, rope, bucket and hand-crank winch	1.45
商人书桌	wooden-writing-desk-with-open-map-book-globe-and-inkpot-with-quill	1.01
小王子本人	young boy character with blue coat, red scarf, beige pants and black shoes	1.55 (绑骨)

核心建议

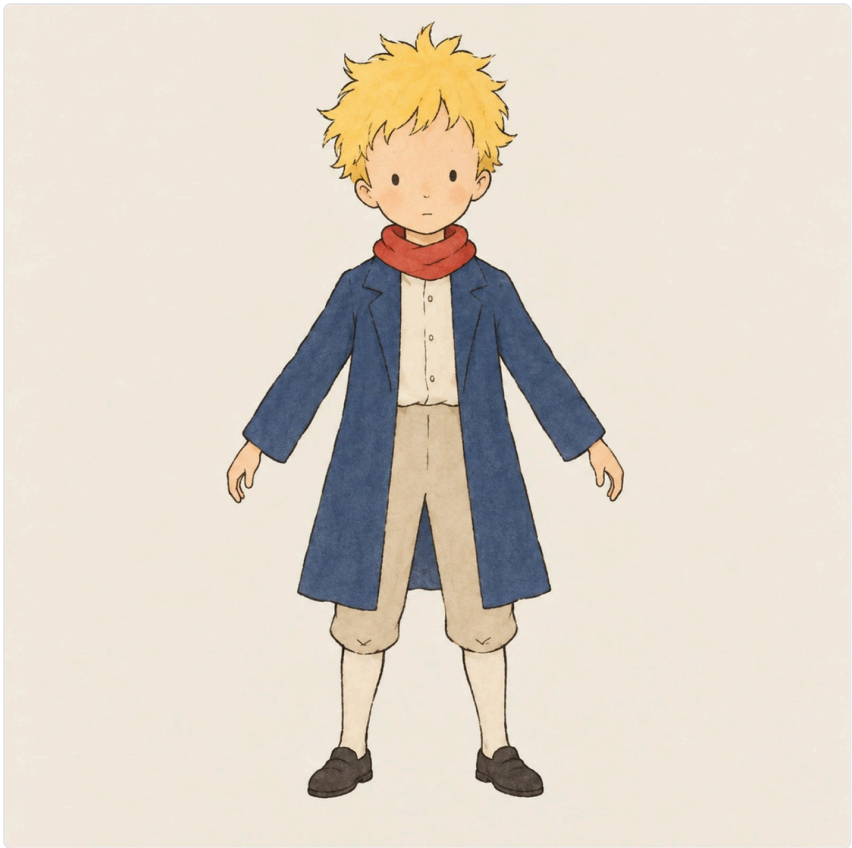
不用渲染，读名字就知道它理解成了什么，这是目前最省事也最靠谱的一道验收。井那一条尤其能说明问题，绳子、木桶、手摇轱辘全在名字里，说明它不是生成了一个「井形物体」，是真把那张图读懂了。

球面本身不走Tripo。星球的几何是代码生成的（一个球加顶点位移），球面数学自己算更准；但地表长什么样交给Codex，两张2048×1024的等距圆柱贴图，白天版是暖赭石地表配鼠尾草绿地块，夜版是冷靛蓝。这又是那条分工判据的一次应用：尺寸和拓扑（附录D）归代码，观感归AI。

注意

球面贴图加载时要做一道左右接缝的交叉淡化，不然绕球走一圈，你会看到一条缝。

让主角自己走路



专门为绑骨画的那一张：全身入画、正视角、手臂向外张开与躯干有空隙、两腿分开站稳。四肢糊在一起是绑不出骨架的。



接进引擎之后。41关节的骨架走着Tripo动画库里的行走循环。HUD显示脚下星球直径28米、镜头退到3米外。

绑骨走的是Tripo自带的自动绑定，20积分。**硬约束 § 06讲了全套**——骨架预设要从默认的「适用于动物」换成「适用于类人角色」、第一次失败重试也扣20、导出带两条动画按下标取会取错、位移轨要切、朝向差90°。这个项目里咬过我的就是前两条。

贴图性能那一课：先量，再动手

第一版跑起来，反馈是「过于卡顿了」「性能需要优化500%」。我没有直接去减面，先量了一次，量出来的结论和直觉完全相反。下面这几张表里的毫秒数全部是浏览器侧的读数，和 § 20说的那个「算子耗时」是两码事，别混着读：一个量的是你的网页每帧花多久，一个量的是Tripo那边模型算多久。

量到的	数值
整条后期链GPU耗时	0.37 ms（场景+阴影+泛光+调色，520万像素）
主线程JS	1.5 ms / 帧
三角面/绘制调用	147k-285k / 25-56
60 fps的预算	16.7 ms

渲染链只用掉了预算的八分之一。那卡顿从哪儿来？把场景里所有材质的贴图列出来，答案立刻出现：31张唯一贴图，其中8192²的10张、4096²的20张，估算显存5.0 GiB（这是按1024进位的读数，换成本书算显存的十进制口径是5.38 GB，两边是同一笔账的两种进位，口径见 § 12）。

Tripo默认导出8192²的PBR贴图（基于物理的渲染贴图组，底色、法线、金属粗糙度各一张，附录D）。一个道具在画面里才一两米高、屏幕上几百个像素，却挂着一张268 MB的底图（8192²按RGBA32展开，加上mipmap约358 MB）。17个资产加起来5.0 GiB，GPU一直在做纹理的换入换出，表现出来就是卡。

处理方式是只降贴图、不碰几何，一条命令：

```
gltf-transform resize 产出/x.glb 网页/assets/tripo/x.glb --width 1024 --height 1024
```

用 `resize` 而不是 `optimize`，后者默认会连减面一起跑，而几何根本不是瓶颈，没有理由动它。主角给2048²（他是特写对象），其余1024²。

结果是贴图显存5.0 GiB降到0.22 GiB，23倍；GLB载荷89.87 MiB降到12.30 MiB，7.31倍；三角面一个没动。逐项对照表、以及「当天记录写的97.6 MB我没能复现」这件事怎么处理的，都在 § 12那一节。

画质在这个观看距离上肉眼无损，王座的木纹红绒金冠、主角的脸和扣子，特写下都还在。

核心建议
这一轮真正的教训：卡顿的第一反应是「少画点东西」，但先量一次GPU耗时，可能会发现GPU是闲的。这次要是直接去减面、关泛光、砍阴影，画质掉一大截，而真正吃掉5.0 GiB的那批贴图一张没动，问题原封不动。

玫瑰的表演：把一段文本做成一套机制



A观的一天：遇到蛇、提水、给玫瑰浇水、日落、拉远看尺度、夜里。画面底部那行法语是走近某个元素时浮出来的原文，配一句我自己的中文直译。

原作第八章其实是一份角色机制说明书，它把动机都写明了：她因为被拆穿一个天真的谎而恼羞，于是咳嗽，好让他觉得是自己不对。而这一章的结论就是这套玩法本身，我本该照她的行为判断她，而不是照她的话。

做成机制就是：你看着的时候，她把花瓣撑起来装没事；你一走开，垂才回来。

状态	花瓣下垂量
没人在看	0.501
你走近	0.111（撑起来了）

没人在看（离 $8.1 > 7.5$ ）· 花瓣下垂 0.501你走近了（离 $6.5 < 7.5$ ）· 撑起来了 0.110

同一朵玫瑰，同一口渴值（0.911），差别只在玩家站在哪。左：站在8.11米外，她不知道有人看，花瓣下垂 $\text{rotation.x} = 0.5013$ ；右：走到6.46米，越过7.5米那条线，她「装没事」0.6秒后撑起来，只剩0.1103。这套机制刁钻的地方就在这儿——你走近才看得清，可一走近她就撑起来了。拍这两张时把咳嗽和夜间抖动按住了（机制本身一行没改），否则量到的不是纯下垂量。2026-09-11实拍。



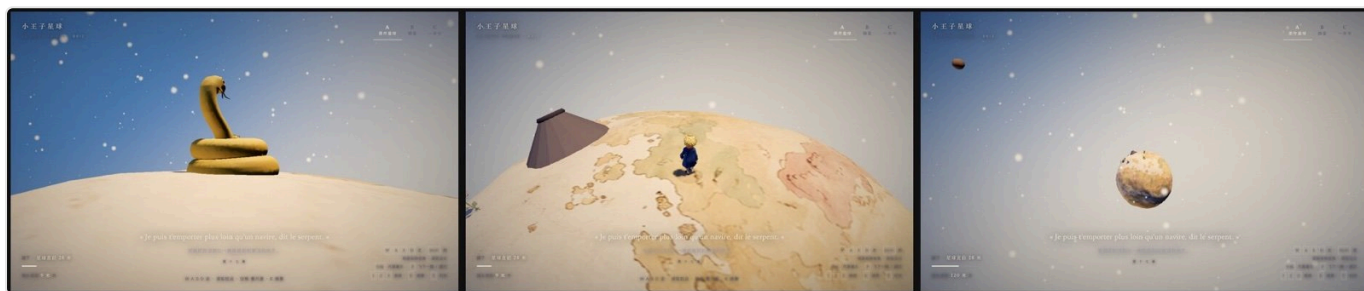
玫瑰的特写。所以判断她说没说谎的唯一办法是：靠近、再退开、看她。咳嗽也照原文做了真假两种，假的比真的更使劲（实测峰值 0.83对0.43）。

台词全部逐字取自第八章，一句没编，四条触发规则：渴（渴到一定程度只有一半概率说实话）、冷、被拆穿、久不来看她。

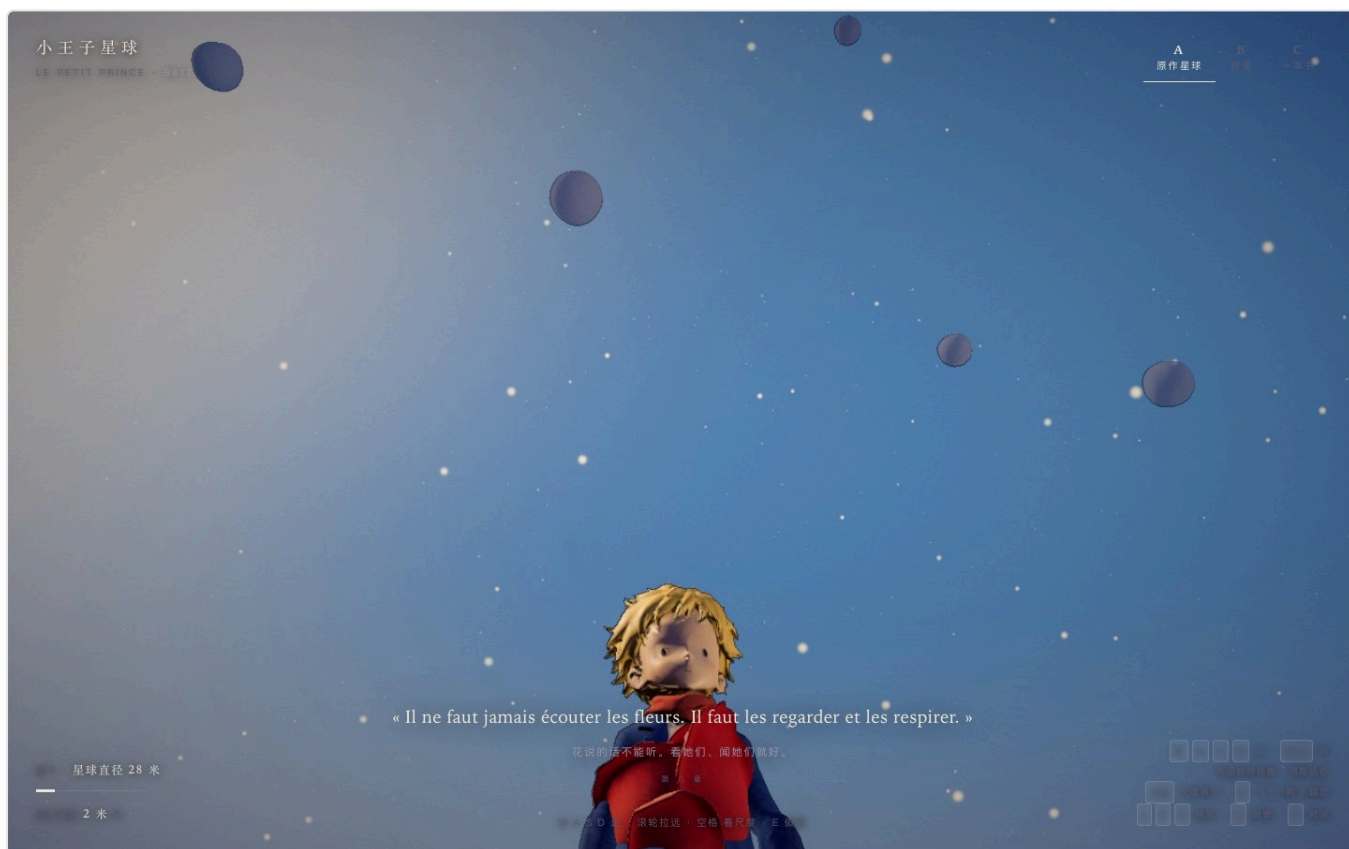
这一节可以直接搬走的方法：先在原始文本里找出「动机已经被写明」的段落，那就是机制的设计稿。表演的参数不用你发明，作者写过了。

整个A方向的地基只有一句话：**做一件事等于把这一天的时钟往前拨**。六个动词（拔树苗、打水、浇水、通火山、陪她说话、罩玻璃罩）全做一遍要1.29个单位，而一个白天只有0.56。一天绝对做不完所有事，取舍不是一条规则，是天空自己算出来的，因此也不需要任何进度条，太阳的位置就是那个进度条。

抬头那一刻，和一个相机的难题



按住空格看尺度：从蛇的近景一路退到130米外，整颗星球变成星尘里的一个点。规模对比是这个主题的核心表现手段。



抬头那一幕的实拍帧：站在玫瑰旁边（lat 8 / lon 6），镜头拉到滚轮下限、俯仰压到-0.95。同一帧里能数出五颗别的星球，第六颗在画面左上角外。配的是第八章原句：« Il ne faut jamais écouter les fleurs. Il faut les regarder et les respirer. »2026-09-11实拍。

别的星球从第一帧就挂在天上，游戏永远不提示它们。你之所以直到某一天才看见，是因为在那之前你一直低着头做那六件事。实测（站在玫瑰旁边扫八个朝向）：平视一颗都看不见，俯仰压到-0.35冒出一颗，抬到

-0.95、再把镜头拉到最近，扫一圈总共能找到六颗——但同一帧里最多只装得下五颗，第六颗永远在画面外。这个区别值得说一句：六是「转一圈能看全几颗」，五才是「你一眼能看见几颗」，我一开始把两个数记混了。

注意

纯轨道相机没法「抬头」，这是这一段真正的难点。相机永远看向主角，想抬头就只能把相机压到主角下方，在球面上那就是钻进星球内部（试过，画面里能看到自己星球上的井和椅子从里侧飘着）。正解是把视线从相机位置里解耦：相机位置只用俯仰角的非负那一半，负的那一半拿去单独抬视线。

三个静默的bug，全都是画面看不出来的



道具自检出图。七件道具接进引擎那一轮，看着「模型都在」，一量才发现三个都是静默错的。这张图是修好之后逐个瞬移取景拍的。

其一，道具埋进地里。加载时有一句 `grp.position.y -= bb.min.y` 把模型底面挪到 $y=0$ ，但克隆时写的是 `c.position.y = y`（赋值），把这个偏移抹掉了。实测王座埋了0.950米、路灯1.025米、商人的书桌反过来悬空0.430米。改成 `+=` 就好：一个字符的bug，只在把模型摆到地上那一刻才显形。

其二，一行动画把整张书桌拎走了。本来只想让抽屉里锁着的星星转一转，写的是遍历 `children` 里所有 `isGroup` 的节点，而Tripo模型正是作为一个Group挂进书桌的，于是整张桌子跟着转起来、被拎到0.9米高。修法是给星星那一组起名字、点名操作。一个容器里既有你的零件又有外来的模型，按类型遍历就是定时炸弹。

其三，主角的身高从来就没对过。第一版我以为解决了，收敛出一组很精确的数字，实际渲染出来的人只有0.79米，和1.01米的书桌摆在一起比桌面还矮。根因是四处测量问题叠在一起：量的是世界Y的极值（人一摆到

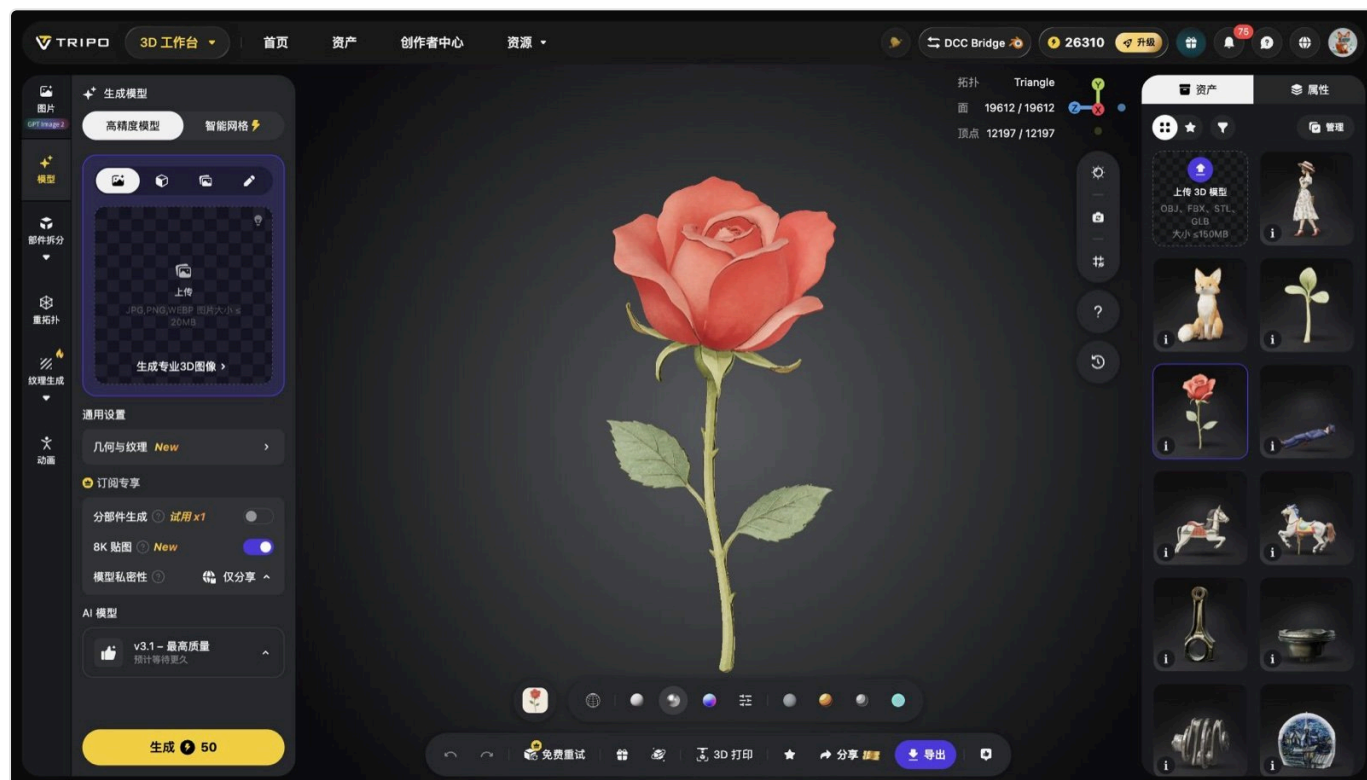
球面上就是斜的)、蒙皮矩阵没刷新、于是在一组假数据上做了一次正确的拟合。三处的完整拆解和修法在 § 06。

注意

量骨骼网格的时候，量法本身就是被测对象的一部分。这类bug可怕在它全链条自证：拟合收敛、代码不报错、模型和骨架都是好的，连「贴地0.064米」这种细节都算得出来。修好之后一次收敛到1.547，而最初 `normalize()` 给的1.55一直都是对的。

这三条的共同点是：画面只会告诉你「这东西看着不对」，告诉你「埋了0.95米」的是数。所以我写了一个无头浏览器的自检脚本，逐个道具瞬移取景、量真高和离地间隙、出图。有它之前，我靠肉眼从截图上估比例，连续判错了三次。

还有一次，我把好模型判成了坏模型



Tripo自己的预览页里的玫瑰：花头、花萼、花茎、叶片全在，是一朵完全正确的玫瑰。而我在自己的夜景demo里把它判成了「一团皱巴巴的红块」。

我曾经判定玫瑰和狐狸都重建失败了，两次都是错的。错在读图的方式：我是在夜景、低角度、贴得很近的截图里判的，那种条件下低面数模型会退化成剪影。更关键的是，我没有先去看Tripo自己的预览就下了结论，而那是免费、直接、权威的一手证据。

核心建议

验收资产的正确姿势：中性光、正面、相机退到能看见整体，或者直接看Tripo预览；比例和落点只能从包围盒（附录D）和离地间隙里读，不能从截图里估。顺带纠正我当时写错的一条归纳，「细长结构是图生3D的弱项」没有证据，那朵玫瑰就是一根细长花茎，重建得干干净净。

这一颗的账

项	积分
Tripo生成16次（8个元素+ 7个道具+主角）	800
自动绑骨20 × 2（第一次失败要重试）	40
合计	840积分 ≈ ¥39.2

三个能走进的世界、17个资产、一个会走路会表演的主角，不到四十块钱。花掉的小时数才是大头，而那些小时几乎全用在两件事上：**搭一条能重复跑的流水线，和一套能揪出静默错误的自检。**

附录A 资源包索引

Appendix A · Resource Kit Index

这本书用到的全部模型的逐件数据：多少面、几张贴图、有没有骨骼、对应哪张设计图、当初那句提示词怎么写的、花了多少积分。要复现哪一件，从这张表开始查。

核心建议

这张表里的面数和顶点数，是直接解GLB的索引数组数出来的，不是抄Tripo界面的读数。两条通道在个别件上会差一点：比如 car-01-taxi，Studio右上角报的是18,497个三角面 / 12,522个顶点，这张表写的是18,509 / 12,770，差12个面、248个顶点（§ 01那一节也说破了这处差异）。拿这张表当验收单逐件对的时候，以文件读数为准——你自己解一遍GLB，数出来的会是表里这组。

注意

先说清楚哪些你能直接拿到。这份索引分两半，别记混：**A到F这六组共56件、81.1 MiB，是2026-09-11装配进随书资源包的那一批**，逐件核过三角面、顶点、贴图张数与尺寸、骨骼、动画、体积，拷贝后SHA256与源文件一致，你解开zip就能用。**G、H、I这三组是09-12新做的P2.0批次**，当时资源包已经封好了，所以它们目前**不在包里**。索引照样逐件写全，是因为这批的数据本身就是这一轮最有信息量的东西（尤其是「同一张图换个模式会怎样」）。要不要把它们补进包里另说，但我不想让你照着表去解压一个不存在的文件。

先说这些数字是怎么来的。不是抄Tripo界面上显示的，是用脚本直接读每个GLB文件的12字节头和后面的JSON块算出来的，贴图尺寸从二进制块里的PNG / JPEG文件头取，不装任何三方库、全程只读，没有改动过任何源文件。所以这些数字是文件本身的数字，和你把同一个文件拖进Blender或three.js量出来的应该一致。

体积这一列按1024进位，单位写作MiB（1 MiB等于1,048,576字节）。拿它对文件的时候：Windows资源管理器也按1024进位，读数和这里对得上；macOS的访达按1000进位，同一个文件它显示的数会比这里大4.9%，不是拷贝出了问题。全书正文讲贴图与显存那笔账用的是另一个口径——十进制MB，1 MB当100万字节（§ 12写了为什么），差的也是这4.9%。看单位就能分辨：写MiB的是这份索引里的文件体积，写MB的是正文那笔账。下面G组那段把同一批文件的两套数并排放了一次。

大多数组给两张表：前一张是几何与贴图，后一张是它从哪张设计图、哪句提示词来的（G组和H组因为输入图都还在Tripo账号里能原样取回，后一张表改成列生成参数与积分）。几个列要先说清楚（这几列里的术语——贴图、ORM、绑骨、关节——附录D都用人话讲过一遍）：

- **三角面/顶点**：glTF accessor里的真实计数。纯动画文件没有网格，两列都是「—」
- **贴图（张数 × 最大尺寸）**：这一列必须连着生成模式一起读，两种模式给的东西不一样。**v3.1（高精度模型）**出三张：基色、法线、金属粗糙度（ORM），所以A到F这六组全是「3 ×」。P2.0（智能网格）只出一张基色，没有法线图、也没有ORM，所以G和H两组全是「1 ×」。这不是导出时漏勾了什么，是模式本身的差别，后果是同一盏灯下P2.0的件会比v3.1的哑一点，接场景之前值得在真实光照里先看一眼（§ 12讲怎么处理）。列里给的是张数和其中最大的那张的分辨率

- **绑骨/动画**：关节数+动画片段名。全资源包只有E组这三件有骨骼
- **提示词**：这批提示词都是「风格：…。画面内容：…」的两段式，风格那段一组里共用，单独引在表前；表格里给的是画面内容那段的逐字前60字，后面用省略号截断。完整提示词在资源包的 jobs.jsonl 里

核心建议

A组和B组放在一起看最有信息量：同样19件，体积从26.4 MiB掉到9.8 MiB。先说清楚这一对不是「只降贴图」的纯对照——B组是拿A组的文件在本地跑 gltf-transform optimize 降下来的，那条命令默认连一遍轻度简化一起跑，所以三角面也从345,935降到了338,924 (-2.0%)。把16.6 MiB的降幅拆开看：贴图字节从15.9 MiB降到2.6 MiB，省了13.3 MiB；几何和其余字节从10.5 MiB降到7.2 MiB，省了3.3 MiB（那2%的面加上顶点量化一起贡献的）。**大头在贴图**。要一个几何完全没动过的对照，看下面的G组：38件合计735,232个三角面，降贴图前后逐件相等。这也是为什么书里反复说，模型跑不动先去看贴图，别急着减面。

A • 巴黎街景原件（1K贴图档）• 19件 • 26.4 MiB

来源项目是**巴黎街景 • 原件**（调研/实测记录/assets/），整个资源包的主干。全套走「Codex出设计图 → Tripo图生3D」这条流水线，设计图和提示词都在本地，没有第三方素材混进来。正文里巴黎那批说的是23件，那是最终进游戏的总数；这里的19件是其中留了设计图和提示词、你能照着复现的那一批。19件里9件用的是下面这句风格前缀打底，另外三种变体是角色版5件、去掉配色括号的简写版3件、载具版2件：

风格：低多边形写实卡通风游戏资产原画，奥斯曼巴黎复古配色（奶油色石材/藏青灰蓝屋顶/黑色锻铁/黄铜细节），暖调柔和摄影棚布光，纯浅灰色背景无阴影无文字水印，3/4等距透视单个主体居中构图，细节清晰适合 image-to-3D重建。

文件名	类别	三角面	顶点	贴图（张数 × 最大尺寸）	绑骨/动画	体积	来源
building-01-corner.glb	建筑	17,215	16,534	3 × 1024 × 1024	—	1.80 MiB	assets/
building-02-tall-attic.glb	建筑	17,366	14,294	3 × 1024 × 1024	—	1.82 MiB	assets/
building-03-boulangerie.glb	建筑	16,874	16,814	3 × 1024 × 1024	—	1.49 MiB	assets/
building-04-fleuriste.glb	建筑	17,428	16,442	3 × 1024 × 1024	—	1.53 MiB	assets/
building-05-kiosque.glb	建筑	18,441	14,878	3 × 1024 × 1024	—	1.44 MiB	assets/

文件名	类别	三角面	顶点	贴图 (张数 × 最大尺寸)	绑 骨/ 动 画	体积	来源
building-06-arc.glb	建筑	16,711	14,421	3 × 1024×1024	—	1.42 MiB	assets/
building-07-sacre-coeur.glb	建筑	18,976	14,698	3 × 1024×1024	—	1.46 MiB	assets/
building-08-metro.glb	建筑	18,117	16,818	3 × 1024×1024	—	1.49 MiB	assets/
building-09-bookstall.glb	建筑	17,195	15,148	3 × 1024×1024	—	1.45 MiB	assets/
car-01-taxi.glb	载具	18,509	12,770	3 × 1024×1024	—	1.19 MiB	assets/
car-02-cargo-tricycle.glb	载具	18,941	15,238	3 × 1024×1024	—	1.40 MiB	assets/
pedestrian-01-man-suit.glb	人物	19,594	12,105	3 × 1024×1024	—	1.04 MiB	assets/
pedestrian-02-woman-dress.glb	人物	19,510	12,291	3 × 1024×1024	—	1.21 MiB	assets/
pedestrian-03-cyclist.glb	人物	19,287	15,495	3 × 1024×1024	—	1.48 MiB	assets/
pedestrian-04-dog-walker.glb	人物	19,358	12,358	3 × 1024×1024	—	1.10 MiB	assets/
pedestrian-05-painter.glb	人物	19,026	13,013	3 × 1024×1024	—	1.12 MiB	assets/
prop-01-cafe-table.glb	道具	19,380	13,236	3 × 1024×1024	—	1.35 MiB	assets/
prop-02-bench.glb	道具	16,828	11,804	3 × 1024×1024	—	1.17 MiB	assets/

文件名	类别	三角面	顶点	贴图（张数 × 最大尺寸）	绑 骨/ 动 画	体积	来源
-----	----	-----	----	------------------	-------------------	----	----

prop-03-planter.glb	道具	17,179	19,445	3 × 1024 × 1024	—	1.40 MiB	assets/
---------------------	----	--------	--------	--------------------	---	----------	---------

同样这19件，每一件是从哪张设计图、哪句提示词来的：

文件名	对应设计图	提示词（画面内容前60字）
building-01-corner.glb	building-01-corner.png	一栋巴黎奥斯曼风格转角公寓建筑，弧形立面转角，浅灰色石材外墙，底层拱形商铺橱窗，上层带铸铁阳台栏杆，顶部灰蓝色锌板孟莎屋…
building-02-tall-attic.glb	building-02-tall-attic.png	一栋巴黎奥斯曼风格公寓建筑，藏青色百叶窗，六层高带阁楼老虎窗，米色石材外墙，底层石砌基座。
building-03-boulangerie.glb	building-03-boulangerie.png	一间巴黎街角面包店建筑门面，红色帆布雨棚，金色手写体招牌字体位置留白，橱窗摆放法棍面包造型剪影，深木色门框。
building-04-fleuriste.glb	building-04-fleuriste.png	一间巴黎花店建筑门面，墨绿色帆布雨棚，门口两侧摆放鲜花木箱与花束（低多边形简化造型），白色石材门面。
building-05-kiosque.glb	building-05-kiosque.png	一座巴黎经典报刊亭（Kiosque à journaux），深绿色铸铁圆顶结构，八边形亭身，顶部小尖塔装饰，复古报纸杂志…
building-06-arc.glb	building-06-arc.png	一座巴黎凯旋门风格纪念拱门，米黄色石材，单拱结构，拱顶浮雕装饰简化处理，厚重石柱基座。
building-07-sacre-coeur.glb	building-07-sacre-coeur.png	一栋巴黎圣心大教堂风格建筑，纯白色石材外墙，标志性白色圆顶，罗马式拱窗。
building-08-metro.glb	building-08-metro.png	一座巴黎经典地铁站入口（Guimard风格），祖母绿色铸铁曲线花纹拱门框架，黄色Metropolitain字样招牌位置留…

文件名	对应设计图	提示词（画面内容前60字）
building-09-bookstall.glb	building-09-bookstall.png	一间巴黎塞纳河畔风格旧书店门面，墨绿色木质门面，门口书摊木箱陈列旧书（低多边形简化造型），铜制门把手细节。
car-01-taxi.glb	car-01-taxi.png	一辆巴黎复古出租车，雷诺4CV造型，黑色车身搭配奶油黄车顶，车顶装有TAXI字样灯箱位置留白。
car-02-cargo-tricycle.glb	car-02-cargo-tricycle.png	一辆巴黎街头送货三轮车（vélo cargo），前置木质货箱，深绿色车身。
pedestrian-01-man-suit.glb	pedestrian-01-man-suit.png	一位站立的巴黎绅士角色，深蓝色西装，手提棕色公文包，微笑站姿。
pedestrian-02-woman-dress.glb	pedestrian-02-woman-dress.png	一位站立的巴黎女士角色，碎花连衣裙，手挎藤编购物篮，优雅站姿。
pedestrian-03-cyclist.glb	pedestrian-03-cyclist.png	一位骑着巴黎经典城市自行车的角色，棕色复古自行车配车篮，角色穿休闲装。
pedestrian-04-dog-walker.glb	pedestrian-04-dog-walker.png	一位牵着小型犬散步的老年角色，呢子大衣，手持牵引绳，小狗低多边形简化造型。
pedestrian-05-painter.glb	pedestrian-05-painter.png	一位塞纳河畔街头画家角色，坐在画架前作画姿态，贝雷帽，画架与调色盘细节。
prop-01-cafe-table.glb	prop-01-cafe-table.png	一组巴黎露天咖啡馆藤编桌椅组合，红棕色藤编椅两把配小圆桌，桌上放咖啡杯碟（低多边形简化造型）。
prop-02-bench.glb	prop-02-bench.png	一张巴黎公园长椅，深绿色木条座椅配黑色铸铁扶手支架。
prop-03-planter.glb	prop-03-planter.png	一个巴黎街头石质花坛，米色石材方形花坛，内种植低矮灌木与花卉（低多边形简化造型）。

B · 同上19件的512贴图减面版 · 19件 · 9.8 MiB

来源项目是巴黎街景 · 减面版（调研/实测记录/assets/optimized/），和A组一一对应：同一批Tripo任务的产物，B组是A组的文件在本地又过了一道 gltf-transform optimize（贴图降到512²，顺带跑了这条命令默认的轻度简化，所以三角面比A组少2.0%）。风格前缀与A组完全相同，所以下面只给数据表和来源表。

文件名	类别	三角面	顶点	贴图 (张 数 × 最大 尺寸)	绑 骨/ 动 画	体积	来源
building-01-corner.glb	建筑	17,043	16,453	3 × 512×512	—	0.60 MiB	assets/ optimized/
building-02-tall-attic.glb	建筑	17,177	14,202	3 × 512×512	—	0.59 MiB	assets/ optimized/
building-03-boulangerie.glb	建筑	16,696	16,723	3 × 512×512	—	0.55 MiB	assets/ optimized/
building-04-fleuriste.glb	建筑	17,232	16,341	3 × 512×512	—	0.56 MiB	assets/ optimized/
building-05-kiosque.glb	建筑	18,101	14,710	3 × 512×512	—	0.52 MiB	assets/ optimized/
building-06-arc.glb	建筑	16,619	14,375	3 × 512×512	—	0.50 MiB	assets/ optimized/
building-07-sacre-coeur.glb	建筑	18,734	14,578	3 × 512×512	—	0.52 MiB	assets/ optimized/
building-08-metro.glb	建筑	17,849	16,679	3 × 512×512	—	0.55 MiB	assets/ optimized/
building-09-bookstall.glb	建筑	16,961	15,030	3 × 512×512	—	0.54 MiB	assets/ optimized/
car-01-taxi.glb	载具	18,083	12,555	3 × 512×512	—	0.48 MiB	assets/ optimized/
car-02-cargo-tricycle.glb	载具	18,493	14,999	3 × 512×512	—	0.55 MiB	assets/ optimized/
pedestrian-01-man-suit.glb	人物	19,042	11,831	3 × 512×512	—	0.43 MiB	assets/ optimized/
pedestrian-02-woman-dress.glb	人物	18,602	11,841	3 × 512×512	—	0.46 MiB	assets/ optimized/

文件名	类别	三角面	顶点	贴图（张数 × 最大尺寸）	绑骨/动画	体积	来源
pedestrian-03-cyclist.glb	人物	19,105	15,401	3 × 512×512	—	0.57 MiB	assets/optimized/
pedestrian-04-dog-walker.glb	人物	18,530	11,942	3 × 512×512	—	0.44 MiB	assets/optimized/
pedestrian-05-painter.glb	人物	18,542	12,771	3 × 512×512	—	0.45 MiB	assets/optimized/
prop-01-cafe-table.glb	道具	18,746	12,919	3 × 512×512	—	0.50 MiB	assets/optimized/
prop-02-bench.glb	道具	16,320	11,545	3 × 512×512	—	0.44 MiB	assets/optimized/
prop-03-planter.glb	道具	17,049	19,382	3 × 512×512	—	0.58 MiB	assets/optimized/

同样这19件，每一件从哪张设计图、哪句提示词来——与A组逐件相同，见上面那张来源表（B组就是A组的文件又过了一道 gltf-transform optimize，来源当然没变）。

C · 巴黎扩展 · 植被与街具 · 6件 · 8.7 MiB

来源项目是巴黎扩展 · 原件（调研/实测记录/assets-2/），补齐A组缺的树和街道家具。三棵梧桐是全资源包唯一「减面减不动」的一类，E1实测下限95.3%，配 S 04读。6件里3件用下面这句植被专用前缀，2件沿用A组那句主句，剩下1件（鸽群）把配色约束整段去掉了：

风格：低多边形写实卡通风游戏植被资产原画，奥斯曼巴黎复古配色，暖调柔和摄影棚布光，纯浅灰色背景无阴影无文字水印，3/4视角单棵树完整居中构图，树干与树冠比例正确，细节清晰适合image-to-3D重建。

文件名	类别	三角面	顶点	贴图（张数 × 最大尺寸）	绑骨/动画	体积	来源
prop-04-wallace-fountain.glb	道具	18,816	12,281	3 × 1024×1024	—	1.31 MiB	assets-2/

文件名	类别	三角面	顶点	贴图（张数 × 最大尺寸）	绑骨/动画	体积	来源
prop-05-morris-column.glb	道具	17,712	11,905	3 × 1024×1024	—	1.15 MiB	assets-2/
prop-06-pigeon-flock.glb	道具	19,864	13,885	3 × 1024×1024	—	1.31 MiB	assets-2/
tree-01-plane-green.glb	植被	18,414	20,490	3 × 1024×1024	—	1.57 MiB	assets-2/
tree-02-plane-autumn.glb	植被	18,380	21,663	3 × 1024×1024	—	1.84 MiB	assets-2/
tree-03-trimmed-ball.glb	植被	19,122	21,608	3 × 1024×1024	—	1.52 MiB	assets-2/

同样这6件，每一件是从哪张设计图、哪句提示词来的：

文件名	对应设计图	提示词（画面内容前60字）
prop-04-wallace-fountain.glb	prop-04-wallace-fountain.png	一座巴黎华莱士喷泉（Fontaine Wallace），深绿色铸铁圆亭造型，四根女像柱支撑圆顶，顶部小尖塔，中段圆形水盆…
prop-05-morris-column.glb	prop-05-morris-column.png	一根巴黎莫里斯柱（Colonne Morris）广告柱，深绿色铸铁圆柱，顶部圆锥形小圆顶带尖饰，柱身环绕海报展示框，底部…
prop-06-pigeon-flock.glb	prop-06-pigeon-flock.png	一小群巴黎街头鸽子，五只灰蓝色鸽子聚在一处，其中两只展翅姿态、三只站立啄食，造型低多边形简化但可辨识。
tree-01-plane-green.glb	tree-01-plane-green.png	一棵巴黎街头梧桐树，高大笔挺的浅灰色斑驳树干，茂密宽大的掌状绿色树冠，树冠饱满有层次。
tree-02-plane-autumn.glb	tree-02-plane-autumn.png	一棵巴黎街头梧桐树，秋季金黄色树叶，浅灰色斑驳树干，树冠通透有疏密层次，部分枝叶稀疏。
tree-03-trimmed-ball.glb	tree-03-trimmed-ball.png	一棵巴黎街边修剪成整齐球形树冠的行道树，深绿色紧密圆整的树冠，笔直树干，法国园艺风格。

D · 游乐设施8K原档 · 6件 · 30.8 MiB

来源项目是**游乐园 · 游乐设施原件**（[游乐园.../unity-验证/游乐设施/产出/](#)）。皮克斯道具设定图的路子，和巴黎那套是两个美术方向，可以直接对比同一套流水线换个风格前缀会出什么。这组是没优化过的8192²原档，留着当「不优化长什么样」的对照组，六件加起来30.8 MiB，比A + B + C加起来还大。

它的提示词是「皮克斯动画电影的道具设定图：〈主体〉。单个物体、完整入画、居中放在纯白背景上，四分之三视角，只有极淡的接触阴影、无其他物体、无文字、无人物。均匀柔和的摄影棚布光，颜色鲜明干净。」这种前后夹心式，六件共用首尾，表里给的是中间那段主体描述。

文件名	类别	三角面	顶点	贴图（张数 × 最大尺寸）	绑骨/动画	体积	来源
carousel.glb	机械	19,048	13,253	3 × 8192×8192	—	5.75 MiB	游乐设施/产出/
ferris-cabin.glb	机械	19,405	12,510	3 × 8192×8192	—	4.82 MiB	游乐设施/产出/
haunted-house.glb	建筑	18,462	12,685	3 × 8192×8192	—	5.90 MiB	游乐设施/产出/
pirate-ship.glb	机械	18,390	12,490	3 × 8192×8192	—	5.58 MiB	游乐设施/产出/
swing-chair.glb	机械	18,707	12,625	3 × 8192×8192	—	5.36 MiB	游乐设施/产出/
teacup.glb	机械	19,393	11,969	3 × 8192×8192	—	3.43 MiB	游乐设施/产出/

同样这6件，每一件是从哪张设计图、哪句提示词来的：

文件名	对应设计图	提示词（画面内容前60字）
carousel.glb	01-旋转木马.png	一整座华丽的旋转木马，金色雕花顶棚、红白条纹的圆形底座、中央立柱，木马清晰可辨。单个物体、完整入画、居中放在纯白背景上，…
ferris-cabin.glb	02-摩天轮吊舱.png	一个复古摩天轮的吊舱，红白双色车身、球形玻璃窗、黄铜色金属吊架。单个物体、完整入画、居中放在纯白背景上，四分之三视角，只…
haunted-house.glb	05-鬼屋.png	一座游乐园鬼屋小楼，紫灰色木板墙、墨绿尖顶、歪斜的窗框和歪掉的烟囱、门口有南瓜灯。单栋建筑、完整入画、居中放在纯白背景上…

文件名	对应设计图	提示词（画面内容前60字）
pirate-ship.glb	08-海盗船.png	一艘游乐园海盗船的船体，深木色船身、金色船首装饰、黑色与金色条纹、两侧各有一排座位。单个物体、完整入画、居中放在纯白背景…
swing-chair.glb	03-飞椅.png	一把游乐园旋转飞椅的吊椅，红白条纹座椅、金属链条从顶部吊下、圆盘形顶盖。单个物体、完整入画、居中放在纯白背景上，四分之三…
teacup.glb	04-旋转茶杯.png	一个游乐园旋转茶杯，粉蓝与奶白配色、圆润的杯身、底部是圆形茶托底座、杯口有描金边。单个物体、完整入画、居中放在纯白背景上…

E • 绑骨与动画 • 3件 • 3.9 MiB

全资源包唯一带骨骼的三件，来源分属**游乐园 • 绑骨实测**和**巴黎扩展 • 原件**两个项目，所以这组的「来源」列不一样，看那一列区分。骨架都是Tripo自动绑骨给的41关节，动画是预设 preset:biped:walk，2.38秒一个循环。注意 pedestrian-01-rigged-walk.glb 有骨架没动画， pedestrian-01-walk-anim.glb 才带动画，**绑骨和动画是两个文件**，这条坑在 § 06展开讲。

文件名	类别	三角面	顶点	贴图（张数 × 最大尺寸）	绑骨/动画	体积	来源
绑骨输出.glb	人物	19,830	11,733	3 × 1024×1024	41关节/ preset:biped:walk	1.27 MiB	绑骨实测/
pedestrian-01-rigged-walk.glb	人物	19,594	12,043	3 × 1024×1024	41关节	1.28 MiB	assets-2/
pedestrian-01-walk-anim.glb	人物	19,594	12,043	3 × 1024×1024	41关节/ preset:biped:walk	1.32 MiB	assets-2/

同样这3件，每一件是从哪张设计图、哪句提示词来的：

文件名	对应设计图	提示词（画面内容前60字）
绑骨输出.glb	T-Pose参考图.png	无
pedestrian-01-rigged-walk.glb	无	无
pedestrian-01-walk-anim.glb	无	无

F • 三模式对照 • 3件 • 1.5 MiB

来源项目是三模式实验 • 模式对照（调研/实测记录/三模式实验/）。同一个源模型分别走Blender减面、命令行直连、Blender手工绑定三条路的产物，单看没什么用，配着 § 13 的实验记录读才有价值。modeA-rigged.glb 那件是反面教材：Blender自动权重只绑了4根骨，实测一转spine整个人就倾倒。

文件名	类别	三角面	顶点	贴图（张数 × 最大尺寸）	绑骨/动画	体积	来源
modeA-blender.glb	建筑	6,772	9,340	3 × 512×512	—	0.36 MiB	三模式实验/
modeA-rigged.glb	人物	19,041	11,850	3 × 512×512	4关节	0.80 MiB	三模式实验/
modeB-direct.glb	建筑	6,781	9,566	3 × 512×512	—	0.36 MiB	三模式实验/

同样这3件，每一件是从哪张设计图、哪句提示词来的：

文件名	对应设计图	提示词（画面内容前60字）
modeA-blender.glb	building-03-boulangerie.png	一间巴黎街角面包店建筑门面，红色帆布雨棚，金色手写体招牌字体位置留白，橱窗摆放法棍面包造型剪影，深木色门框。
modeA-rigged.glb	pedestrian-01-man-suit.png	一位站立的巴黎绅士角色，深蓝色西装，手提棕色公文包，微笑站姿。
modeB-direct.glb	building-03-boulangerie.png	一间巴黎街角面包店建筑门面，红色帆布雨棚，金色手写体招牌字体位置留白，橱窗摆放法棍面包造型剪影，深木色门框。

G • 巴黎P2.0重生（1K贴图档）• 40件 • 52.0 MiB • 不随包

来源目录是P2.0重生 • 1K档（P2.0重生/glb-1024/），2026-09-11到09-12这两天新增的一整批，也是本附录里唯一全部走智能网格P2.0的一组。它和前面A到F最大的不同不在面数，在贴图：P2.0只出一张基色图，没有法线图，也没有ORM，所以下面这张表的贴图列全是「1 ×」，不是前面那些组的「3 ×」。这不是导出时漏了，是这个模式本来就只给一张，处理办法见 § 12。



G组40件排在一起。每一格是同一套灯光、同一个机位渲出来的，三座桥、六座新地标、两条船、公交警车送货车Vélib、七个各不相同的行人都在这张图里。格下的文件名和三角面数在这个尺寸上只能看个大概，逐件的准确数字看下面那张表。渲图与标注由本地脚本批量生成，2026-09-12。

这40件里有38件是各自独立的Tripo任务，另外两件（`tree-01-marronnier-green`、`tree-02-marronnier-autumn`）是拿 `street-06-marronnier-tree` 那一件在本地改贴图颜色做出来的，几何逐位相同、没有再花一分积分。这一手在 § 21 讲了做法。

核心建议

这组的合计数字值得单独记一下：**38件源文件合计509.9 MiB，降完1K贴图是50.0 MiB，只剩9.8%，而三角面合计735,232，逐件一个没动。加上那两件改色树，目录里40个文件共52.0 MiB。省下来的全是贴图。这38件就是 § 12和 § 17那张对照表里的那38件**，那边写的是534.6 MB → 52.4 MB：同一批文件、同一个9.8%，差的4.9%是进位口径——那边按十进制MB，这里按1024进位（MiB），不是两次不同的测量。A组B组那一对指向同一个结论，但这一组才是干净的对照：那一对顺带减掉了2%的面，这38件一个三角面都没动，而且倍数更夸张，因为源头是8192²。

文件名	类别	三角面	顶点	贴图（张数 × 最大尺寸）	绑骨/动画	体积
bridge-01-pont-alexandre-iii.glb	桥梁	32,274	47,752	1 × 1024×1024	—	2.25 MiB

文件名	类别	三角面	顶点	贴图（张数 × 最大尺寸）	绑骨/动画	体积
bridge-02-pont-des-arts.glb	桥梁	30,773	51,379	1 × 1024×1024	—	2.11 MiB
bridge-03-pont-neuf.glb	桥梁	28,159	39,470	1 × 1024×1024	—	1.90 MiB
building-01-haussmann-corner-cafe.glb	建筑	26,231	38,539	1 × 1024×1024	—	1.85 MiB
building-02-haussmann-balcony.glb	建筑	24,712	33,612	1 × 1024×1024	—	1.63 MiB
building-03-art-nouveau-guimard.glb	建筑	29,228	34,591	1 × 1024×1024	—	1.75 MiB
building-04-opera-garnier.glb	建筑	27,120	43,857	1 × 1024×1024	—	2.02 MiB
building-05-louvre-pyramid.glb	建筑	29,718	47,886	1 × 1024×1024	—	2.15 MiB
building-06-pantheon.glb	建筑	28,846	38,821	1 × 1024×1024	—	1.79 MiB
building-07-musee-orsay-clock.glb	建筑	30,311	45,077	1 × 1024×1024	—	2.06 MiB
building-08-moulin-rouge.glb	建筑	28,458	41,617	1 × 1024×1024	—	1.87 MiB
building-09-notre-dame-west.glb	建筑	25,432	40,583	1 × 1024×1024	—	1.89 MiB
building-10-bouquiniste-wall.glb	建筑	26,354	36,427	1 × 1024×1024	—	1.69 MiB
car-03-delivery-van.glb	载具	16,430	20,099	1 × 1024×1024	—	1.09 MiB
car-04-scooter.glb	载具	14,977	15,121	1 × 1024×1024	—	0.82 MiB

文件名	类别	三角面	顶点	贴图 (张数 × 最大尺寸)	绑骨/动画	体积
pedestrian-06-accordionist.glb	人物	15,388	20,707	1 × 1024×1024	—	1.16 MiB
pedestrian-07-child-balloon.glb	人物	15,433	18,474	1 × 1024×1024	—	0.99 MiB
pedestrian-08-couple.glb	人物	15,430	20,303	1 × 1024×1024	—	1.05 MiB
person-01-cafe-waiter.glb	人物	16,897	17,207	1 × 1024×1024	—	0.91 MiB
person-02-police-officer.glb	人物	16,497	18,989	1 × 1024×1024	—	0.99 MiB
person-03-office-woman-baguette.glb	人物	15,753	15,569	1 × 1024×1024	—	0.91 MiB
person-04-beret-elder-dog.glb	人物	16,715	17,785	1 × 1024×1024	—	1.12 MiB
person-05-jogger.glb	人物	16,464	16,586	1 × 1024×1024	—	0.98 MiB
person-06-kid-scooter.glb	人物	16,127	17,538	1 × 1024×1024	—	1.10 MiB
person-07-tourist-camera.glb	人物	16,126	17,486	1 × 1024×1024	—	1.01 MiB
prop-07-icecream-cart.glb	街具	12,175	14,944	1 × 1024×1024	—	0.85 MiB
street-01-bus-shelter.glb	街具	13,458	13,982	1 × 1024×1024	—	0.68 MiB
street-02-velib-station.glb	街具	11,203	12,644	1 × 1024×1024	—	0.69 MiB
street-03-cafe-terrace-set.glb	街具	12,082	14,220	1 × 1024×1024	—	0.98 MiB

文件名	类别	三角面	顶点	贴图（张数 × 最大尺寸）	绑骨/动画	体积
street-04-stone-planter-roses.glb	街具	11,250	17,697	1 × 1024×1024	—	1.16 MiB
street-05-iron-planter-geranium.glb	街具	10,970	16,204	1 × 1024×1024	—	0.99 MiB
street-06-marronnier-tree.glb	植被	11,825	19,770	1 × 1024×1024	—	1.17 MiB
tree-01-marronnier-green.glb	植被	11,825	19,770	1 × 1024×1024	—	1.01 MiB
tree-02-marronnier-autumn.glb	植被	11,825	19,770	1 × 1024×1024	—	1.06 MiB
vehicle-01-ratp-bus.glb	载具	11,819	13,251	1 × 1024×1024	—	0.77 MiB
vehicle-02-bateau-mouche.glb	载具	15,375	24,257	1 × 1024×1024	—	1.31 MiB
vehicle-03-peniche.glb	载具	16,022	22,323	1 × 1024×1024	—	1.19 MiB
vehicle-04-citroen-ds.glb	载具	15,924	18,395	1 × 1024×1024	—	0.96 MiB
vehicle-05-paris-police-car.glb	载具	16,568	19,141	1 × 1024×1024	—	1.03 MiB
vehicle-06-velib-bike.glb	载具	16,708	20,929	1 × 1024×1024	—	1.09 MiB

同样这40件，每一件是怎么生成出来的。这一组的来源表和前面几组不一样：P2.0的输入图和提示词都还在Tripo账号里，随时能按uuid取回原图，所以这里改成列**Tripo自己给模型起的名字**（那是它对这张图的理解，最便宜的一道验收）、**面数上限填了多少**、**Tripo读回来的四边面数**，以及**贴图那一步花了多少秒**。耗时是算子耗时口径，不含上传与排队。

文件名	Tripo起的名字	face_limit 设定	四边面 读数	贴图 算子 耗时	积分（生 成+贴图）
bridge-01-pont-alexandre-iii.glb	ornate bridge 3d model	15,000	20,289	92s	65+30
bridge-02-pont-des-arts.glb	stone bridge 3d model	15,000	20,389	139s	65+30
bridge-03-pont-neuf.glb	stone arch bridge 3d model	15,000	17,402	128s	65+30
building-01-hausmann-corner-cafe.glb	parisian building 3d model	15,000	15,941	199s	0+30
building-02-hausmann-balcony.glb	historic building 3d model	15,000	14,899	170s	65+30
building-03-art-nouveau-guimard.glb	ornate building 3d model	15,000	16,907	200s	65+30
building-04-opera-garnier.glb	historic opera house 3d model	15,000	16,005	233s	65+30
building-05-louvre-pyramid.glb	louvre pyramid 3d model	15,000	18,332	200s	65+30
building-06-pantheon.glb	cathedral 3d model	15,000	15,996	200s	65+30
building-07-musee-orsay-clock.glb	clock tower 3d model	15,000	17,468	241s	65+30
building-08-moulin-rouge.glb	red windmill 3d model	15,000	16,171	548s	65+30
building-09-notre-dame-west.glb	gothic cathedral 3d model	15,000	14,507	138s	65+30
building-10-bouquiniste-wall.glb	rustic book boxes 3d model	15,000	16,162	98s	65+30
car-03-delivery-van.glb	vintage van 3d model	8,000	9,368	157s	65+30

文件名	Tripo起的名字	face_limit 设定	四边面 读数	贴图 算子 耗时	积分（生 成+贴图）
car-04-scooter.glb	vintage scooter 3d model	8,000	8,234	157s	65+30
pedestrian-06-accordionist.glb	accordion player 3d model	8,000	8,784	175s	65+30
pedestrian-07-child-balloon.glb	low-poly child figure 3d model	8,000	8,605	115s	65+30
pedestrian-08-couple.glb	romantic couple 3d model	8,000	8,915	163s	65+30
person-01-cafe-waiter.glb	waiter 3d model	8,000	9,256	142s	65+30
person-02-police-officer.glb	police officer 3d model	8,000	9,443	105s	65+30
person-03-office-woman-baguette.glb	woman 3d model	8,000	8,445	118s	65+30
person-04-beret-elder-dog.glb	man with dog 3d model	8,000	9,314	170s	65+30
person-05-jogger.glb	athletic man 3d model	8,000	9,090	105s	65+30
person-06-kid-scooter.glb	scooter 3d model	8,000	8,945	175s	65+30
person-07-tourist-camera.glb	photographer 3d model	8,000	8,705	144s	65+30
prop-07-icecream-cart.glb	ice cream cart 3d model	6,000	6,734	204s	65+30
street-01-bus-shelter.glb	modern bus stop shelter 3d model	6,000	7,443	170s	65+30
street-02-velib-station.glb	key card dispenser 3d model	6,000	6,638	95s	65+30

文件名	Tripo起的名字	face_limit 设定	四边面 读数	贴图 算子 耗时	积分（生 成+贴图）
street-03-cafe-terrace-set.glb	outdoor furniture set 3d model	6,000	6,786	188s	65+30
street-04-stone-planter-roses.glb	stone planter 3d model	6,000	7,379	151s	65+30
street-05-iron-planter-geranium.glb	planter box 3d model	6,000	7,044	112s	65+30
street-06-marronnier-tree.glb	tree 3d model	6,000	8,620	218s	65+30
tree-01-marronnier-green.glb	无独立生成任务：由 street-06-marronnier-tree 本地改色而来，几何逐位相同，0积分				
tree-02-marronnier-autumn.glb	无独立生成任务：由 street-06-marronnier-tree 本地改色而来，几何逐位相同，0积分				
vehicle-01-ratp-bus.glb	bus 3d model	8,000	6,537	103s	130+60
vehicle-02-bateau-mouche.glb	tourist boat 3d model	8,000	9,309	73s	65+30
vehicle-03-peniche.glb	barge 3d model	8,000	8,793	78s	65+30
vehicle-04-citroen-ds.glb	vintage car 3d model	8,000	9,005	122s	65+30
vehicle-05-paris-police-car.glb	police car 3d model	8,000	9,757	112s	65+30
vehicle-06-velib-bike.glb	city bike 3d model	8,000	9,552	143s	65+30

这张表里藏着三件对你有用的事。

一、**face_limit** 是目标，不是硬上限。38件里35件的四边面读数高于填进去的那个数，中位数是设定值的1.13倍，最狠的一件是那棵七叶树：填6000，出来8620，1.44倍。所以你要卡一个严格的面数预算，得把设定值往下压一档，别照着预算填。

二、四边面转三角面的倍数，这一批实测在1.37到1.87之间，中位数1.75。合计431,169四边面变成735,232三角面，整批平均1.705倍。这个数不是2。只有纯四边形网格才趋近2，而生成出来的网格里总混着三角面和多边面。\$ 04那张对照表里的两个数（8,596→15,431是1.795，5,615→9,800是1.745）落在这个分布里面。做面数预算时用1.75去估，比用2稳。

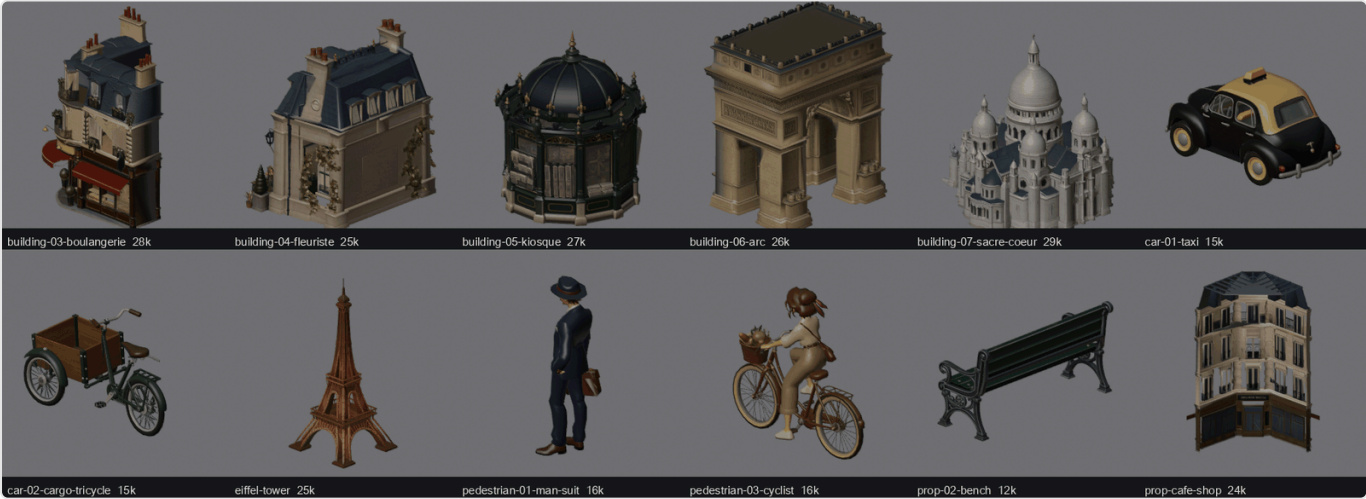
三、贴图那一步比生成慢得多。网格那一步Tripo界面上实测三十几秒到一分钟，贴图这一步逐件读数是73秒到548秒，中位在150秒上下。红磨坊那件548秒是全场最慢的。做批量计划时，真正决定你几点能收工的是贴图这一栏，不是网格那一栏。

注意

公交车那一行的 130+60 不是打错。 vehicle-01-ratp-bus 提交了两次：第一次面数上限根本没生效，出来16,881个四边面，只好照8000重做一遍，所以生成和贴图各付了两笔。为什么没生效在\$ 01讲了（输入框的值进了DOM但没进前端框架的状态），那也是本轮最值钱的一个坑。另外那一行 0+30 是Preview期的首次免费试用，只免了生成那一笔，贴图照付30；免完之后落回的是65这个划线折扣价。

H · P2.0补生成（含第一件P2.0绑骨行人）· 13件 · 19.2 MiB · 不随包

来源目录是P2.0重生成 · 新生成1K档（ P2.0重生成/glb-1024-新/ ）。G组那38件是从账号里已有的批次里盘出来的，这13件是2026-09-12当天现做的：把G组没覆盖到的槽位补齐，外加一件全书唯一的P2.0绑骨行人。书里E组那三件绑骨资产全部是v3.1， pedestrian-01-walk.glb 是第一件走智能网格出来、再自动绑骨、再套行走动画的。



H组里12件静态资产（第13件是绑骨行人，带骨架，不适合放进这种静态渲图）。铁塔那件是本轮重做的：旧件渲出来是紫蓝色的塔，新件是暖褐铜色，Tripo自己给它起的名字是 wooden eiffel tower ，说明它把这个暖褐读成了木质，颜色方向对了。渲图与标注由本地脚本批量生成，2026-09-12。

文件名	类别	三角面	顶点	贴图（张数 × 最大尺寸）	绑骨/动画	体积
building-03-boulangerie.glb	建筑	28,303	41,723	1 × 1024×1024	—	1.96 MiB

文件名	类别	三角面	顶点	贴图（张数 × 最大尺寸）	绑骨/动画	体积
building-04-fleuriste.glb	建筑	25,473	39,583	1 × 1024×1024	—	1.85 MiB
building-05-kiosque.glb	建筑	27,574	41,072	1 × 1024×1024	—	1.86 MiB
building-06-arc.glb	建筑	26,737	34,558	1 × 1024×1024	—	1.65 MiB
building-07-sacre-coeur.glb	建筑	29,587	44,442	1 × 1024×1024	—	1.97 MiB
car-01-taxi.glb	载具	15,431	17,652	1 × 1024×1024	—	1.00 MiB
car-02-cargo-tricycle.glb	载具	15,970	21,548	1 × 1024×1024	—	1.12 MiB
eiffel-tower.glb	建筑	25,102	36,590	1 × 1024×1024	—	1.66 MiB
pedestrian-01-man-suit.glb	人物	16,126	19,696	1 × 1024×1024	—	1.02 MiB
pedestrian-01-walk.glb	人物	16,178	17,888	1 × 1024×1024	41关节/ preset:biped:walk	1.30 MiB
pedestrian-03-cyclist.glb	人物	16,023	21,249	1 × 1024×1024	—	1.13 MiB
prop-02-bench.glb	道具	12,752	15,562	1 × 1024×1024	—	0.85 MiB
prop-cafe-shop.glb	建筑	24,495	38,744	1 × 1024×1024	—	1.81 MiB

这13件的来源。「面数上限」那一列里凯旋门有两行，第一行是本轮唯一一次浪费：面数上限填了15000，提交时实际按默认5000走了，只好重做一件顶替，旧的作废（详见 § 01那条输入框的坑）。耗时是**网格算子耗时**，不含上传与排队：

文件名	Tripo起的名字	输入图	face_limit设定	网格算子耗时
building-03-boulangerie.glb	parisian bakery storefront	tripo-source/building-03-boulangerie.png	15,000	47秒
building-04-fleuriste.glb	flower shop storefront	tripo-source/building-04-fleuriste.png	15,000	45秒
building-05-kiosque.glb	ornate newsstand	tripo-source/building-05-kiosque.png	15,000	62秒
building-06-arc.glb (作废件)	arc de triomphe	tripo-source/building-06-arc.png	填15,000，实走5,000	17秒
building-06-arc.glb (采用件)	grand arch	tripo-source/building-06-arc.png	15,000	未记录
building-07-sacre-coeur.glb	cathedral	tripo-source/building-07-sacre-coeur.png	15,000	50秒
car-01-taxi.glb	vintage car	tripo-source/car-01-taxi.png	8,000	22秒
car-02-cargo-tricycle.glb	wooden cargo bike	tripo-source/car-02-cargo-tricycle.png	8,000	21秒
eiffel-tower.glb	wooden eiffel tower	参考图/eiffel-tower-ref.png (本轮新画)	15,002	31秒
pedestrian-01-man-suit.glb	stylized gentleman	tripo-source/pedestrian-01-man-suit.png	8,000	22秒

文件名	Tripo起的名字	输入图	face_limit设定	网格算子耗时
pedestrian-01-walk.glb	vintage businessman	参考图/pedestrian-walk-tpose-ref.png (本轮新画, A-Pose)	8,000	未记录
pedestrian-03-cyclist.glb	girl on bicycle	tripo-source/pedestrian-03-cyclist.png	8,000	26秒
prop-02-bench.glb	park bench	tripo-source/prop-02-bench.png	6,000	17秒
prop-cafe-shop.glb	corner cafe building	参考图/prop-cafe-shop-ref.png (本轮新画)	15,000	65秒

核心建议

pedestrian-01-walk.glb 是这一组里最值得单独看的一件。16,178三角面、1副骨架、1024²单张基色、1.30 MiB，输入是一张**A-Pose**（手臂下斜30到40度）的设计图，**自动绑定19秒第一次就成**。落盘校验直接读FBX里的字符串：41个Cluster、1个Skin、82个LimbNode。它和E组那三件v3.1绑骨件放在一起对比最有价值：同一条绑骨流程，换了生成模式，贴图从三张变成一张，网格也更轻（它顶替掉的那件v3.1绑骨行人是19,007三角面、三张贴图）。

另有两件注意事项，不写进表里会让你白跑一趟：**P2.0出的绑骨件导出时带了两条动画轨**（一条空的 `ArmatureAction`，一条真正的 `preset:biped:walk`），代码里按下标取第一条会取到空的，得按名字挑；它的**正面朝+X**，而v3.1那批行人朝+Z，装进同一个场景要先转90度。这两条都在 § 06和 § 12展开。

I · P2.0的8K内嵌贴图原档 · 15个FBX · 约150 MiB · 不随包

来源目录是**P2.0重生成 · 新生成原档**（P2.0重生成/新生成/）。这是H组那13件在降贴图之前的样子：Tripo的智能网格配上贴图之后，`model_url` 给的是FBX不是GLB，贴图**8192²内嵌在文件里**。留着它有一个具体用途，当「**不降贴图长什么样**」的对照组。D组那六件8K原档回答的是同一个问题，但那是v3.1的三张8K；这一组是P2.0的单张8K，两组放在一起，你能看清「贴图张数」和「贴图尺寸」各自贡献了多少体积。

文件	体积	说明
building-03-boulangerie.fbx	14.17 MiB	下面12件同理：带8192 ² 内嵌基色贴图的成品原档
building-04-fleuriste.fbx	12.43 MiB	

文件	体积	说明
building-05-kiosque.fbx	12.78 MiB	
building-06-arc.fbx	15.67 MiB	
building-07-sacre-coeur.fbx	15.18 MiB	
car-01-taxi.fbx	8.18 MiB	
car-02-cargo-tricycle.fbx	9.19 MiB	
eiffel-tower.fbx	10.46 MiB	
pedestrian-01-man-suit.fbx	6.99 MiB	
pedestrian-03-cyclist.fbx	8.47 MiB	
prop-02-bench.fbx	9.50 MiB	
prop-cafe-shop.fbx	12.00 MiB	
pedestrian-01-walk.fbx	7.12 MiB	绑骨之前的纯网格版，不是会走路的那个
pedestrian-01-walk_rigging.fbx	7.39 MiB	网格+41关节骨架
pedestrian-01-walk_retargget.fbx	0.68 MiB	只有骨架和行走动画，没有网格，必须和上一件配对用

注意

最后这三件的命名很容易骗人。不带后缀的 pedestrian-01-walk.fbx 是绑骨之前的纯网格，看文件名以为它会走路，装进去就是个木头人。要会走的那个人，得同时拿 _rigging 和 _retarget 两件。这条和E组那两件 v3.1绑骨资产的坑是同一条，只是这里连文件名都在误导你，所以单独标出来。

小王子星球 · 19件 · 只收设计图与提示词

注意

这一组不随包发GLB。《小王子》原著在多数司法辖区已进入公有领域，但角色形象在多国仍有活跃的商标与形象权登记，「过了版权期」这一句挡不住随书公开分发模型本体的风险。所以资源包里只放 输入/ 那22张设计图和5个 jobs*.jsonl 提示词，当一套完整的图生3D流水线示例用。下面这张表列出它们本来生成了什么，方便你自己跑一遍对结果。

这一组是全项目提示词最系统的一批：同一句插图风格前缀打底，逐件换画面内容。19件里15件用下面这句：

风格：安托万·德·圣埃克苏佩里《小王子》插图的语言——深色钢笔线条勾出干净轮廓，内部平涂低饱和水彩色块，暖米白纸面质感，形体概括、块面分明，没有复杂纹理、没有高光渐变，适合image-to-3D重建。纯浅米灰背景，无地面投影、无文字、无水印。

剩下4件（小王子本人那几个文件）用的是绑骨专用变体，多加了「画面里没有复杂纹理、没有高光渐变、**没有细长垂挂物**，适合image-to-3D重建与**自动骨骼绑定**」。这里要提醒一句：当时写这条约束是因为我以为「细长结构是图生3D的弱项」，**后来证明这条归纳是错的**——那朵玫瑰就是一根细长花茎，重建得干干净净，我是在夜景低角度的截图里误判的。资源包里的 01b-玫瑰-短茎大花头 和 02b-猴面包树苗-短粗 就是当时为这个误判重画的备用设计图，最后没送Tripo。

另外这组有个副产品值得单独说：19件里有16件Tripo给出了自动命名，而且**全部命中**。比如 well.glb 被命名为「stone well with wooden frame, rope, bucket and hand-crank winch」，水井、木架、绳子、桶、手摇绞盘五个部件一个不落。读模型名就知道Tripo把你的输入理解成了什么，不用渲染，这是最便宜的一道验收。

文件名	类别	三角面	顶点	贴图（张数 × 最大尺寸）	绑骨/动画	体积
baobab-sprout.glb	植被	19,582	11,703	3 × 8192×8192	—	7.43 MiB
baobab.glb	植被	19,176	11,460	3 × 8192×8192	—	7.30 MiB
biz-desk.glb	道具	16,612	10,748	3 × 8192×8192	—	4.95 MiB
chair.glb	道具	17,343	10,741	3 × 8192×8192	—	5.87 MiB
crate.glb	道具	16,864	11,033	3 × 8192×8192	—	7.96 MiB
drunk-table.glb	道具	19,582	12,146	3 × 8192×8192	—	5.87 MiB
fox.glb	人物	19,240	12,810	3 × 8192×8192	—	4.64 MiB
geo-desk.glb	道具	16,956	10,515	3 × 8192×8192	—	6.97 MiB
lamp.glb	道具	19,422	12,524	3 × 8192×8192	—	4.94 MiB
mirror.glb	道具	19,586	11,648	3 × 8192×8192	—	4.40 MiB
prince-anim.glb	人物	—	—	无	41关节/ preset:biped:walk	0.13 MiB
prince-rig.glb	人物	19,842	13,269	3 × 8192×8192	41关节	4.81 MiB
prince-walk.glb	人物	—	—	无	41关节/ preset:biped:walk	0.13 MiB

文件名	类别	三角面	顶点	贴图（张数 × 最大尺寸）	绑骨/动画	体积
prince.glb	人物	19,844	13,266	3 × 8192×8192	—	4.76 MiB
rose.glb	植被	19,612	12,197	3 × 8192×8192	—	4.08 MiB
sheep.glb	人物	19,698	11,465	3 × 8192×8192	—	2.59 MiB
snake.glb	人物	19,154	11,346	3 × 8192×8192	—	2.70 MiB
throne.glb	道具	16,528	10,572	3 × 8192×8192	—	7.17 MiB
well.glb	道具	19,318	12,096	3 × 8192×8192	—	8.07 MiB

文件名	对应设计图	提示词（画面内容前60字）
baobab-sprout.glb	02-猴面包树苗.png	一株刚发芽的猴面包树小苗，细细的茎上顶着三片圆钝的叶子，还没有木质化的树干，看起来人畜无害。
baobab.glb	03-猴面包树成株.png	一棵成年的猴面包树，树干粗壮膨大像一只倒扣的瓶子，顶端伸出七八根像树根那样分叉的粗枝，没有细枝也没有叶子，体量占满画面高…
biz-desk.glb	34-商人的书桌.png	一张深色木质的办公书桌，桌面上摊开一本厚厚的暗红色账本，旁边放着一只带铜锁扣的木质钱箱，箱盖微微掀开一条缝。
chair.glb	37-椅子.png	一把朴素的木质靠背椅，四条细腿，座面是平的木板，靠背由两根立柱和三根横档组成，没有任何装饰。
crate.glb	36-箱子.png	一只方方正正的木板箱，箱子有盖子，盖子上开着一排圆圆的小通气孔，箱子侧面用木板钉出的横竖纹理清楚。
drunk-table.glb	33-酒瓶和桌.png	一张矮矮的圆木桌，桌面上凌乱地摆着五六只深绿色玻璃酒瓶，两三只立着、两三只横倒着，还有一只歪在桌沿。
fox.glb	04-狐狸.png	一只安静坐着的狐狸，橘棕色皮毛，白色的下巴和肚皮，两只尖耳朵，深色的四只爪子，一条粗大的尾巴盘在身体侧面。
geo-desk.glb	35-地理学家的书桌.png	一张宽大的木质书桌，桌面上摊开一本巨大的地图册、书页平展，桌角立着一个小地球仪（架在铜色圆环支架上），旁边斜插着一支白色…
lamp.glb	09-老式路灯.png	一盏老式的铸铁煤气路灯，细长的柱子，顶部是一只六角形的玻璃灯罩，灯罩外有装饰性的铸铁托架和一个小小的尖顶盖。

文件名	对应设计图	提示词（画面内容前60字）
mirror.glb	32-镜子.png	一面立在细木杆上的圆镜，镜框是薄薄的金色圆环，镜面是浅灰蓝色，底座是一个小小的圆盘。
prince-anim.glb	10-小王子-绑骨用.png	一个全身站立的小男孩，头顶是蓬松的金色短发，脸上只有两个小小的墨点当眼睛；穿一件**及膝**的深蓝色长外套，外套下摆不超…
prince-rig.glb	10-小王子-绑骨用.png	一个全身站立的小男孩，头顶是蓬松的金色短发，脸上只有两个小小的墨点当眼睛；穿一件**及膝**的深蓝色长外套，外套下摆不超…
prince-walk.glb	10-小王子-绑骨用.png	一个全身站立的小男孩，头顶是蓬松的金色短发，脸上只有两个小小的墨点当眼睛；穿一件**及膝**的深蓝色长外套，外套下摆不超…
prince.glb	10-小王子-绑骨用.png	一个全身站立的小男孩，头顶是蓬松的金色短发，脸上只有两个小小的墨点当眼睛；穿一件**及膝**的深蓝色长外套，外套下摆不超…
rose.glb	01-玫瑰.png	一株独立的玫瑰花，挺直的花茎上顶着一朵由五六片圆润花瓣围成的花，花萼微微翘起，茎上两片小叶子。
sheep.glb	06-绵羊.png	一只圆滚滚的绵羊，卷曲的米白色羊毛，小小的脑袋，四只短短的腿，安静地站着，表情憨厚。
snake.glb	05-蛇.png	一条盘成两圈的蛇，身体是均匀的金色，头从盘起的身体中央抬起来，尾巴收在盘圈的下方。
throne.glb	31-王座.png	一张高大的木质王座，靠背很高，座面和靠背铺着暗红色的绒布，椅背顶端镶着一顶小小的金色王冠，扶手方直。
well.glb	08-水井.png	一口石头砌的圆口水井，井台由粗糙的浅灰石块垒成，井口上方立着两根木柱，木横梁中间吊着一只木桶，柱子旁边有摇柄绞盘。

哪些件没有设计图或提示词，为什么

资源包不是完美的。有三处缺口，说清楚比藏着好，免得你照着表去找一个根本不存在的文件。

一、E组那两件行人，设计图和提示词都没有。pedestrian-01-rigged-walk.glb 和 pedestrian-01-walk-anim.glb 的三角面（19,594）和 pedestrian-01-man-suit 完全一致，但GLB内部的Tripo任务ID是三个不同的值，没法确认它们是「man-suit绑骨后重导出」还是「另生成的一个T-Pose模型」。想复现就按 S 06的流程自己跑一遍，别指望照着提示词重来。

二、绑骨输出.glb 有输入图，但生成那张图的提示词丢了。它的输入是 T-Pose参考图.png，图还在，当时生成这张图的那句prompt没有逐字留档，记录里只剩一句「真名锚定用Mixamo」。整条绑骨链路每一步都记全了，只有起点那句话没留住，这是我这轮最可惜的一个疏忽。

三、有几件你可能觉得眼熟的，故意没放进来。雪铁龙2CV、埃菲尔铁塔、低模奥斯曼楼、街角咖啡馆这四件是最早直接在Studio网页里点出来的，没走本地流水线，输入图和提示词都没留，你拿到也复现不了；另外2CV是在产商标车型，铁塔的夜间灯光造型在法国另有著作权主张。发动机的活塞连杆曲轴、旋转木马两件、名画那四件属于图生3D，输入本来就是照片或抠图，不存在生图提示词。发动机和木马那两组的输入照片，维基图页上标的是公有领域，那是我当时拿它生成模型的依据；名画那四件的输入是高清复制品的裁切，扫描那一层另有版权。但「自己拿它生成一个模型」和「把成品随书发给几千个人」是两件事，后者的标准更高——我没有逐张回核许可证的具体版本与附加条件，所以这一类在再分发这一层剔除，不代表当初用它有问题（同一口径见 § 00）。

四、G组和H组这批P2.0件，缺的是另外几样东西。输入图和提示词一件不缺（都还在Tripo账号里，按uuid调详情接口就能原样取回，不用重做设计图），缺的是三项：逐件的网格生成耗时，平台没给逐件读数，界面上实测是三十几秒到一分钟这个区间，只有H组那13件因为是现场一件件做的才留下了秒数；那两件改色树的独立任务记录，它们本来就没有独立任务，是拿同一件在本地改的贴图；以及32件的朝向角——朝向是「相对某个旧槽位」的量，这32件还没分配槽位，就没有对照物，所以给不出数，表里干脆不设这一列，别去找。凡是我核不到的，这份表里写「未记录」，不写一个估出来的数。

还有一个反方向的缺口，对你反而是好事：有9张设计图和提示词都齐了，Tripo那一步没跑。巴黎扩展那批的冰淇淋车、手风琴艺人、抱气球的小孩、情侣、送货车、小摩托六张，游乐设施的售票亭和甜品车两张，小王子那架老式飞机一张。资源包里图和词都在，这是成本最低的练手材料。同样的输入，你跑出来的结果和书里那批放在一起比，就知道你手上这个版本的Tripo和2026年9月的差在哪。

（另有两张星球表面图 20-星球表面-等距圆柱、21-星球表面-夜 本来就是贴图不是模型输入，不算缺口。资源包清单里把 01b、02b 记成「最终用了哪一版没有记录」，回查小王子那个项目的记录可以确认：这两张根本没送进Tripo，玫瑰和猴面包树苗用的都是原版 01 和 02。）

附录B 踩坑索引

Appendix B · Pitfall Index

这里收的全是踩过并且修好的，估算和推测一条都不进。碰到症状先在这几张表里查，对得上就直接照修法做，要看来龙去脉再回正文对应的那一节。出处那一列标「本附录」的，正文没有展开过，全部内容就在那一行里。这几张表里的词密度比正文高，看到不认识的（meshopt、ORM、包围盒、实例化池这些）直接翻附录D，那里按你会撞上它们的顺序讲了一遍人话。

一、Three.js与加载GLB

症状	根因	修法	出处
白底下模型造成一片死黑，深色底却正常	Tripo的PBR材质金属度高，漫反射被抑制，亮度全靠反射环境；场景没有环境贴图	用canvas画一张带暗部的渐变当全景环境贴图，走 PMREMGenerator 赋给 scene.environment （代码见本附录末）；别开ACESFilmic色调映射	§ 12 § 20
整个画面发白、没有重量感	环境贴图从白到浅灰，金属反射回来全是亮色	给环境贴图一块暗部，相当于摄影棚的黑旗	§ 20
模型被放大到画面里只剩两只鞋	Box3.setFromObject 量 SkinnedMesh量不准——不是「它拿绑定姿势」（three.js从r155起已按当前姿势算），是那个包围盒缓存不会自己更新，姿势变了你量到的还是上一帧	直接读GLB里 POSITION accessor 的 min / max 拿真实尺寸	§ 12
一排角色像仪仗队，动作完全同步	带skeleton的模型用普通 clone() 会共享骨骼	改用 SkeletonUtils.clone() ，相位用 (x*0.37+z*0.11)%duration 错开。注意这个工具不在精简版 vendor目录里，要补对应版本	§ 12
拆解时某些零件一路飞出屏幕	偏移在每帧的位置计算里累加；静态件不会被重置，被别处每帧赋值的件（气门）加了也白加	偏移只在渲染那一刻生效： asmApply() → render() → asmRestore()	§ 20

症状	根因	修法	出处
改了 <code>material.transparent</code> 没反应	需要重新编译shader	加 <code>needsUpdate = true</code> 。同理 <code>SkeletonHelper</code> 被网格挡住时要 <code>depthTest=false</code> <code>+ renderOrder=999</code>	本附录
自建几何一旋转就被甩变形	形状是相对别的参照点定义的	先移到支点再旋转，形状在自己的 局部坐标里画	本附录
拖了滑块没反应，看着像交互坏了	异步竞态，模型还没加载完，控件 事件白丢	所有模型load完之后，把当前控件 值补应用一次	本附录
后台标签页截图拿到旧画布	<code>requestAnimationFrame</code> 在隐藏 页会被暂停，Chrome对定时器还有 密集节流	本地自校验加 <code>setInterval(frame, 340)</code> 兜 底；要自动化截图直接走无头 Chrome	§ 12
两个零件互相穿进去	把存在几何约束的两个量当成独立 参数各设了一条曲线	先找出约束方程，让其余参数从它 推出来。判据：两个量之间有物理 或几何约束，就不能各设一条曲线	本附录
算出来的指标和领域常识相反	简化模型撑不住那个物理量	先怀疑自己的模型，换一个能逐项 验算的纯几何指标，落点通常更强	本附录
某个零件还没装，画面上却 已经飘着	独立mesh没归进任何一件	凡是 <code>scene.add</code> 的东西都要有归属	§ 20
连调四轮材质都「还是发白」	页面的加载遮罩没淡出，盖在整个 画面上	截图自校验前先确认遮罩消失，自 检副本里把它设成 <code>display:none</code> 。判据：整体发白 且各元素受影响一致，先查覆盖层	§ 20
headless渲染命令什么都没干	macOS没有 <code>timeout</code> 命令，静默 变成command not found	换别的超时方式；跑之前先 <code>pkill</code> <code>-f "headless=new"</code> 清掉堆积的 实例	§ 12
headless Chrome报FAIL， 以为截图失败	它写完文件不退出，命令超时报 错，但文件已经写出来了	收尾按文件是否存在重新盘点，别 信退出码	§ 12
GLB加载被CORS挡住	页面是用 <code>file://</code> 打开的	起一个 <code>python3 -m</code> <code>http.server</code> 。纯静态、无外部资 源的HTML不受此限	§ 21

症状	根因	修法	出处
同一个GLB，这个工具量出来1米，那个工具量出来三万多米	文件用了 <code>KHR_mesh_quantization</code> ，顶点坐标是 <code>normalized:true</code> 的 <code>int16</code> ；不认这个标志的解析器把原始整数直接当世界坐标读，于是高度1.0被读成32767	按分量类型归一化再读。症状很好认：出现32767或它的倍数就是这个，我那次的自动缩放算出了32783的系数	§ 12
近处的树凭空消失，远处的树好好画着	实例池配额满了就 整棵不画 ，没有降级分支。同一个写法在树、烟囱、路灯三处都出现过	抢不到近景坑位就退到远景池画简化版，不是不画。顺带把配额调大：实测三角面144万涨到244万，draw call还是77， 几乎不花钱	§ 18
车撞墙后停住了，码表却一直显示2 km/h	<code>speed *= .65</code> 这行有不动点：每帧油门加0.317，乘0.65，解 <code>v = (v+0.317)*0.65</code> 得0.589 m/s，位移是0但速度永远降不到0	别对速度整体乘系数，按墙的法线把速度拆成法向和切向，法向去掉、切向留一部分。改完撞墙会卸力、贴着墙还能滑出去	§ 18
街上十几个行人长得一模一样	生成函数 <code>makeWalker(key, ...)</code> 的第一个参数从头到尾没被用过，条件分支无条件命中同一支，18个行人全是同一具模型	这类bug画面上看着「有人在走」，一切正常。判据是把实际用到的资产名打出来对账，别看画面	§ 18
把相机摆到地标前截图，八个里七个「什么都没有」	游戏循环每帧用跟车相机覆盖 <code>world.camera</code> ，你设的机位下一帧就没了，截到的是玩家当前朝向的随机视角	改车不改相机 ：把车头转向目标，等跟车相机自己插值到位再截。线索是那次有一个地标居然拍对了，不一致本身就是「测法错了」的信号	§ 18
帧时间方差大得离谱，中位8.57 ms、p95 61.8 ms、最小3.19 ms	另一条命令行正在用无头浏览器跑录制，在抢同一块GPU	量性能前先确认没有别的进程在用GPU。最小值往往还是可信的（那次min 3.19和干净测得的3.29吻合），但那是推断不是测量，别写进结论	本附录
时间轴的标签和横条对不上	刻度用 <code>flex:1</code> 平分，而真实跨度不是均分的	按 <code>(t-T0)/SPAN</code> 绝对定位；挑一两个已知点核对	本附录

唯一一段建议直接抄走的代码（白底3D的环境贴图，不需要任何外部HDR文件）：

```
function studioEnv(rd){
  const c=document.createElement('canvas'); c.width=128; c.height=128;
  const g=c.getContext('2d'), grd=g.createLinearGradient(0,0,0,128);
  grd.addColorStop(0,'#ffffff'); grd.addColorStop(.34,'#e8e4da');
  grd.addColorStop(.52,'#928d82'); grd.addColorStop(.75,'#5c574e');
  grd.addColorStop(1,'#403c35');
  g.fillStyle=grd; g.fillRect(0,0,128,128);
  g.fillStyle='rgba(255,255,255,.9)'; g.beginPath(); g.ellipse(40,24,26,13,0,0,Math.PI*2);
  g.fill();
  const t=new THREE.CanvasTexture(c); t.mapping=THREE.EquirectangularReflectionMapping;
  const pm=new THREE.PMREMGenerator(rd); const env=pm.fromEquirectangular(t).texture;
  pm.dispose(); t.dispose(); return env;
}
scene.environment = studioEnv(renderer);
```

二、Unity (10条，外加三条追加的)

症状	根因	修法	出处
东西都对但看着像模型展览，怎么调光都不对	Tripo导出的GLB被归一化到最长边约1.0单位——不是高度（长椅 1.0×0.549×0.413、出租车 0.522×0.570×1.0，撑满1.0的那条边各不相同）， <code>scale = 1</code> 等于把每栋楼当成1米	实例化后量包围盒， <code>s = 目标高度 / bounds.size.y</code> ，再把底面压到地面。宽扁件按最长边归一	§ 14
GLB在磁盘上、 <code>.meta</code> 也生成了，加载却返回null	Tripo CDN上的 <code>_meshopt</code> 变体带 <code>EXT_meshopt_compression</code> ，工程里没装解码器（glTFast本身支持，装 <code>com.unity.meshopt.decompress</code> 即可）。线索只在 <code>.meta</code> 的 <code>reportItems</code> 里	本地先解压： <code>npx @gltf-transform/cli copy in.glb out.glb</code> 。别指望换URL，去掉 <code>_meshopt</code> 是403	§ 14
批处理建场景，新拷进来的GLB找不到	glTFast的导入是延迟的，批处理里没有帧循环等它， <code>AssetDatabase.Refresh()</code> 也不够	同一条命令跑两遍，第一遍触发导入，第二遍资产就在了	§ 14
场景里有个完整的摩天轮，但不在你以为的地方	<code>AddComponent<Light>()</code> 加在root上，改它的 <code>localPosition</code> 就是改root自己	灯挂到子物体上。诊断别靠猜，直接打印 root位置和包围盒	§ 14
轮子炸成一把乱插的棍子	Unity的 <code>Cylinder</code> 轴向是Y，绕Z转 ϕ 之后轴指向 $(-\sin\phi, \cos\phi, 0)$ ，辐条和外圈的90°弄反了	辐条 $\phi = \theta + 90^\circ$ ，外圈 $\phi = \theta$ 。算一遍就清楚，别靠试	§ 14

症状	根因	修法	出处
58盏路灯全开了，地面还是一片黑	每个物体能吃几盏逐像素光，由 <code>QualitySettings.pixelLightCount</code> 决定（Built-in前向渲染，这一档默认4），不是引擎写死的常数；而整块地面是一个物体。调大也不解决：写成8，58盏里照样有50盏进不来（§ 14坑六）	把光池画进地面贴图，灯位和贴图共享同一份坐标；实体点光只照立在地上的东西。切网格会出现光照接缝，不要	§ 14
夜景整个糊成一片白	Bloom阈值0.72，连被照亮的墙面也一起炸	阈值提到0.92，只让真正的高光发光。这条比intensity重要得多	§ 14
自校验看的是旧截图，白看两轮	macOS上中文文件名被NFD归一化，排序后旧图把新图覆盖回去	渲染直接出ASCII文件名，别在中文名上做文章	§ 14
静帧里旋转木马不转、角色是T-pose	编辑器模式渲染不执行 <code>Update</code> / <code>LateUpdate</code>	这不是bug。要带动作的帧得进Play模式录	§ 14
不知道怎么播放GLB里带的行走动画	以为必须建AnimatorController	<code>glTFast</code> 把clip作为子资源挂进来， <code>LoadAllAssetsAtPath</code> 能枚举到，用Playable直接推给Animator。记得 <code>using UnityEngine.Animations;</code> ， <code>PlayableGraph</code> 标 <code>[NonSerialized]</code>	§ 14
构建报 Succeeded，应用启动即崩： <code>level0 is corrupted</code>	批处理里「编译脚本 → AddComponent → SaveScene」同进程跑，Unity把MonoScript内联进了场景文件（没有guid的纯fileID）。中招哪些类是竞态，同一份代码两次构建结果都不同	场景里一个自定义脚本都不放：场景只存纯数据，行为写进manifest，运行时用 <code>[RuntimeInitializeOnLoadMethod]</code> 按层级路径挂上。构建前加闸门，场景一脏就中止	§ 14
对照版「能正常启动」，于是判定问题出在别处	运行日志里有74行 <code>The referenced script is missing</code> ，39个组件丢了37个，它只是窗口弹出来了	验证构建产物，「进程还活着」是最弱的一级证据。三条同时看：日志里的装配计数、应用自己截的图、组件数量对账	§ 14
WebGL构建 165 MB， <code>WebGL.data</code> 一个文件152 MB	<code>glTFast</code> 直接创建 <code>Texture2D</code> ，不走 <code>TextureImporter</code> ，平台贴图压缩从来没生效过；六个设施是8192 ² 贴图	在GLB层面按槽位分尺寸（基础色1024 / 法线512 / 金属粗糙256）。 <code>--pattern</code> 是glob且大小写不敏感， <code>*ORM*</code> 会误伤 <code>Normal</code> ； <code>resize</code> 只缩不放，先做小尺寸pass	§ 14

三、Tripo Studio与平台侧

症状	根因	修法	出处
图上传了，dropzone还是空态，生成按钮点了没反应	文件确实进了 <code><input type=file></code> ，但change事件没送达React，App的state没更新	自己补发 <code>input</code> 和 <code>change</code> 两个事件。排查顺序：先量 <code>files.length</code> ，在input里是事件问题，不在就是根本没投进去（页面已有参考图时 <code>file input</code> 会被移除，要先刷新）	本附录
点完生成，积分没变、页面没跳转，重试一次白花50积分	界面回显有延迟，不等于任务失败	等几秒再去任务列表复查，不要立刻重试	§ 21
绑出来的骨架是错的	AI模型下拉默认是「v2.5适用于动物」	人物必须换成「v1.0适用于类人角色」	§ 06 § 22
第一次自动绑定失败，预览空白	不是固定行为，我一度这么写是错的。 2026-09-10那两次确实都是第二次才成功，但样本只有2；09-12换了一张A-Pose输入图之后，19秒第一次就过了	真正承重的不是「姿势够不够正」，是 四肢和躯干分不分得开 。第一次失败就重试，同时把输入图换成手臂明显离开身体的那种；重试那20积分照样要算进预算	§ 06 § 22
导出的绑骨模型「没有动画」	不是丢了，是分成两个文件： <code>tripo_rigging_*</code> 是网格+41关节骨架， <code>tripo_retargget_*</code> 是骨架+动画剪辑	两个都下，把前者的骨架装进场景、用后者的剪辑驱动， <code>AnimationMixer</code> 按节点名字绑轨道	§ 06
角色走着走着 来回弹一大步	重定向出来的剪辑带位移轨， <code>Hip.position</code> 的Y位移范围1.338	切掉位移轨只留旋转轨，位移归游戏代码管	§ 06 § 22
智能网格生成完是一只灰模型，一点颜色都没有	智能网格出的本来就是无贴图网格，底栏那句「Retry for better results or proceed to Texture」就是在说这件事	再走一次贴图生成，另付30积分。做预算时智能网格要按两笔算：生成一笔、贴图一笔。 Preview期的首次Trial只免生成那一笔，贴图照付30；免完之后落回的是65这个划线折扣价，不是原价100。 界面上把100划掉、实扣65，所以带贴图的完整一件是65+30=95。价格以官网当时标价为准	§ 07

症状	根因	修法	出处
同一张图，今天生成按钮显示65，昨天是50	不是降级也不是涨价，是上一次把Ultra Mesh Quality开着，被表单记住了。弹层顶部那行小字写着We'll automatically save your settings for next time	点生成前先把几何与贴图弹层展开核一遍开关。这个数是拆得开的：65 = 几何15 + 贴图包35 + Ultra 15	§ 07 附录C
17个资产吃掉5.0 GB显存	导出默认给8192 ² 贴图，一张底图268 MB（加mipmap约358 MB）	<code>gltf-transform resize --width 1024 --height 1024</code> 。用 <code>resize</code> 不用 <code>optimize</code> ，后者会连减面一起跑	§ 22
四边面模型在智能拆分里选不了，上传完没反应	智能拆分页上写着「Unavailable for: Quad models, Rigged models」，智能拆分对四边面模型和已绑骨模型都不可用。P2.0（智能网格）默认出的就是四边面	拆件要在三角面模型上做。同一张图想既拆件又要四边面，就生成两版：v3.1三角面那版拿去智能拆分，四边面那版留着当成品	§ 05
离线解析GLB读出来是乱码	CDN上拿到的是 <code>_meshopt</code> 压缩变体	先用 <code>gltf-transform</code> 解开； <code>tripo_retarget_</code> 没有这个后缀，可以直接读	§ 03 § 14
下载签名URL时 <code>&</code> 和 <code>~</code> 被吃掉	在shell里手敲的	交给Python的requests，或者把URL原样传给浏览器的下载工具	§ 03
面数上限明明填了15000，出来的模型却按5000做	用脚本往输入框打字，DOM上的 <code>value</code> 变了，但前端框架的内部状态没收到，提交时读的还是默认值。界面显示15000、任务详情读回5000，两边都「正常」	用原生setter写值，再手动派 <code>input</code> 、 <code>change</code> 、 <code>blur</code> 三个事件。 硬验证 ：设完按一次方向键右，数字变成15002就说明内部值确实是15000，没变就是没进去	§ 01
上一件设好的面数，下一件又变回默认	弹层顶上那句「自动保存您的设置」保的是别的开关，面数不在其中	每一件都重设一遍，别信上一次的残留。批量做之前先想清楚这件事要花多少次重复劳动	§ 01
连着提交第二件时，图片怎么都传不进去	停在任务详情页直接传下一张，那个页面的文件输入框已经被前端清掉了，上传动作会退化成本「拖到页头」然后失败	每提交一件就回一次生成页 。回去时页面会是降级状态（价格显示、模式选择器都不对），传一张图它自己就修好了	§ 01

症状	根因	修法	出处
等按钮上 纹理生成 30 这几个字出现，等到超时也等不到	那个按钮上的文字和价格是两个独立的DOM节点，按整串文本去匹配永远拼不上	别按文本等，改成等页面空闲两轮再按按钮特征点。这类「肉眼看见了、程序找不到」的，先去看那段文字是不是被拆成了两个节点	本附录
一批提交三件，跑到一半整个动作超时中断	三件是十二步操作，实测113秒撞上超时上限	每批两件 ，实测稳定。超时之后 不要直接重试 ，先用任务详情接口把「哪几件其实已经提交成功了」盘清楚，否则会重复下单	本附录
绑骨页的骨架预设下拉，点了选项没反应	它不是原生的 <code><select></code> ，标准的选择操作会被拒绝，直接点选项也不生效	先点开，再用方向键下加回车。 选完之后那层浮层不会自己消失 ，会盖住底下的「自动绑定」按钮，按Esc没用，点页面别处才关得掉	§ 06
套完动画，「导出」按钮从页面上消失了	不是bug，套完重定向之后那个按钮就不在DOM里了	走任务详情接口拿文件： <code>rigging.model_url</code> 是网格加骨架， <code>retarget[0].model_url</code> 是骨架加动画。 接口这条路反而比点导出更短 ，套完动画立刻就能取	§ 06
上传自己的模型去绑骨，先被收了50积分，还弹出一个要你转X/Y/Z的对话框	那50是外部模型的导入费，「找到最佳角度」那个弹层也只在这条路上出现	模型本来就在账号里的话，去绑骨页右侧直接选 ，0积分，也不弹那个对话框。所以顺序是先在Studio生成、再在Studio绑骨，别下载下来又传上去	§ 06
想用官方API批量提交，发现钱包是0积分，可网页上余额明明有两万多	API钱包和Studio订阅是两套独立计费，订阅的积分不会流到API那边	批量生成走网页端，或者单独给API钱包充值。做计划前先确认你要用的那条通路上有没有钱	§ 07

一条没进上面这张表的账。游乐园那批一共提交了7次、出了6件，多出来的那一次是海盗船：它提交了两次，其中一次扣了50积分、产物始终没落到资产列表里，另一次正常出货（成品就是附录A的D组那件 `pirate-ship.glb`）。这件事**只发生过一次、当时没截图、也没向官方提过工单**，既没踩实也没修好，不符合这份索引的准入规则，所以单独写在这儿，不当成已知缺陷读。真正能拿走的是那个习惯：**批量生成前后各记一次余额，收工时按「生成次数 × 单价」把账对一遍**，再回资产列表点一遍件数。两边对不上，当场就知道该去查哪一单，而不是等到收工很久以后。§ 21那张只有总账、没有逐笔的表，就是没做这件事的代价。

四、Blender与格式转换

症状	根因	修法	出处
从Tripo的CDN下回来的GLB，Blender直接打不开	那是 <code>_meshopt</code> 压缩变体，用了 <code>EXT_meshopt_compression</code> 扩展，Blender 4.4不支持	两条路：在Tripo的导出弹层里直接选FBX，或者本地先 <code>npx @gltf-transform/cli copy in.glb out.glb</code> 解开再导入	§ 13
导出时为了省事关掉纹理，结果模型连UV都没有	v3.1这条路上纹理和UV是绑在一起的，关掉纹理出来的GLB只有 <code>POSITION</code> 和 <code>NORMAL</code> ， <code>materials</code> 是0	想事后自己补贴图的，别关纹理，关了就得重跑一次生成。要纯几何做实验可以关，但心里知道这一步不可逆	§ 13
模型渲出来一模一样，片元开销却翻了一倍	<code>use_backface_culling</code> 这个属性的语义是反的：不开它，导出的glTF里 <code>doubleSided</code> 就是 <code>true</code> ，每个面正反都画一遍。批量转的38件全中招	导出前设 <code>mat.use_backface_culling = not</code> 你想要的双面。这个坑不看GLB里的JSON发现不了，画面上没有任何症状，只有帧时间知道	§ 12 § 17
给模型换个颜色，结果整个材质都坏了	把这个模型挂的贴图全改了一遍。法线图和ORM图里的RGB不是颜色，是表面朝向和材质参数，拿去挪色相等于把材质写坏坏了	先从材质里读出哪一张挂在基色槽位上，只改那一张，读不出来就报错退出，不做盲改。P2.0出的件只有一张基色，不会踩到；v3.1那批三张齐全的一改就毁	§ 21
把树叶染成秋黄，连树干也黄了	判定「这是绿色」的色相窗口开得太宽，从71°起算，树皮那种偏黄绿的棕也被卷进去	窗口收到85°起。判据是看你要动的颜色和不想要的颜色在色相环上隔多远，隔得近就把窗口收到它们中间，宁可漏改边缘	§ 21
导入一次FBX，源文件旁边多出一个目录	Blender会在FBX旁边建 <code><文件名>.fbm/</code> 把内嵌贴图解包出来	先把FBX复制进一个临时目录再导入，别在原始素材目录里跑。这条纯粹是卫生问题，但批量跑一遍之后满目录都是它	§ 17

五、发布与打包

症状	根因	修法	出处
每换一批资产，发布包就胖一圈	打包脚本的目录拷贝没排除归档目录，换下来的旧件被整包带走。实测带了11个旧件、约7.9 MB死重	排除规则写进打包脚本，别靠每次记得手工清。顺手把缓存目录、系统隐藏文件、源格式文件一起排掉。实测从63个文件32.1 MB降到52个文件24.9 MB	§ 19

症状	根因	修法	出处
zip解开之后中文目录名是一串乱码	macOS的zip不写UTF-8标志位	要发出去的包，目录名用ASCII。这条和3D没关系，但凡是打包给别人的东西都会撞上	§ 19
脚本在中文路径下报模块找不到	<code>new URL(import.meta.url).pathname</code> 拿到的是百分号编码的串	用 <code>fileURLToPath()</code> 。中文目录名是国内做这行的默认状态，这条迟早撞上	§ 17

核心建议

这五张表里出现频率最高的一类，是「报错不发生、程序不崩、画面看着也对」的静默错误：模型埋进地里0.95米、对照版其实37个组件全丢、量骨骼量到的是测量假象。它们的共同解法只有一条，把关键量打印出来对账，别靠看画面下结论。

附录C 配置与积分参考

Appendix C · Configurations and Credit Reference

官方标价、每个操作在界面上实际扣多少、五种配法各适合什么件，以及我这两周的完整积分流水。
所有价格都是截至2026-09的官方标价，以官网为准，下面每一张表都标了采集时间和来源，你自己对账时以当天页面为准。

一、订阅套餐（2026-09-11采集自官网Pricing页）

官网Pricing页切到CNY、月付视图（图见 § 07）：Pro ¥140/月含3000积分，这是全书里所有「折合多少钱」的换算基准。

套餐	月付	年付	月度积分	官方估算产能	并发	商用
Free	¥0 / \$0	—	200	≈ 13个模型	1	×（公开模型，非商用）
Pro	¥140 / \$20	¥1680 / \$240	3000	≈ 200个（\$0.16 /模型）	10	✓
Max	¥630 / \$90	¥7560 / \$1080	25000	≈ 1660个（\$0.09 /模型）	100	✓
Team *	¥2310 / \$330	¥13860 / \$1980	90000	≈ 6000个	200	✓

截至2026-09官方标价，以官网为准。

TRIPO

ResearchProductsFeaturesSolutionPricingCreator HubResources

Try Tripo Studio

Compare Plans

Get an overview of what's included in each plan and choose the one that suits you best.

Free

Best for Explorers

\$0.0 / Month

Get Free

Pro

Best for Creators

\$20.00 per month
billed yearly
\$20.00 / monthly

Get Pro

Max

Best for Professionals

\$90.00 per month
billed yearly
\$90.00 / monthly

Get Max

Team

Best for Teams & Studios

\$55.00 month / seat
billed yearly
\$110.00 / month / seat

Get Team

Price

Monthly: \$0.00
Yearly: \$0.00

Monthly: \$20.00
Yearly: \$240.00

Monthly: \$90.00
Yearly: \$1080.00

Monthly: \$330.00
Yearly: \$1980.00

Value Proposition

Try Tripo

Start Creating

Create Better

Create Together

Monthly Credits

200

3000

25000

90000

Estimated Output

≈ 13 models

≈ 200 models

≈ 1660 models

≈ 6000 models

Performance & Productivity

Save Up to 50%

官网Pricing页的Compare Plans对照表，上面那张表里的数字全部出自这里。Price 行写得最清楚：Pro Monthly \$20.00 / Yearly \$240.00、Max \$90.00 / \$1080.00、Team \$330.00 / \$1980.00；月度积分200 / 3000 / 25000 / 90000，官方估算产能≈13 / ≈200 / ≈1660 / ≈6000个模型。注意Team卡片上是「\$55.00 month / seat, billed yearly」对「\$110.00 / month / seat」——按席位算，年付单席半价。2026-09-11。

TRIPO

ResearchProductsFeaturesSolutionPricingCreator HubResources

Try Tripo Studio

Processing Priority	Standard	High Priority	Dedicated Priority	Dedicated Priority
Batch Generation	✗	10	30	30
Edit History	1 Day	7 Days	Permanent	Permanent
Free Retry	✗	3	Unlimited	Unlimited
Model Storage	10	Unlimited	Unlimited	Unlimited
Exports	15 (H2.5 only)	Unlimited	Unlimited	Unlimited
Batch Export	✗	✓	✓	✓
Free Pro Refine	✗	✗	3	✗

Privacy & Commercial Rights

Private Models	✗	✓	✓	✓
Commercial Use	✗	✓	✓	✓

Model Access

High Detail Model	✓	✓	✓	✓
Smart Topology Mesh	Limited Access	✓	✓	✓

Creation Features

同一张对照表往下滚，导出与权限那几行。Free那列的Exports 原文是 15 (H2.5 only)，不是完全不能导；同列的Private Models与Commercial Use都是×，Smart Topology Mesh是Limited Access，Free Retry也是×（Pro给3次，Max/Team无限）。顶上那行被导航条压住一半的是Concurrent Tasks：1 / 10 / 100 / 200。2026-09-11。

核心建议

* **Team那一行的年付别拿计算器去对，它不是月付×12。**Pro和Max都是老老实实的×12

($\$20 \times 12 = \240 、 $\$90 \times 12 = \1080)，Team写的 $\$1980$ 却只相当于×6。差别在它是按席位算的：**月付\$110一席，年付\$55一席，年付本身就是半价。**把席数代进去两条路都能对上——月付 $\$110 \times 3 \text{席} = \330 ，年付 $\$55 \times 3 \text{席} \times 12 = \1980 。顺带把「3席起售」这件事也验了：从月付除得3，从年付除也得3，两条各自独立的算式指向同一个数，而官网页面上从头到尾没有明写过这个最低席数。

注意

看定价页有个坑：四张套餐卡片顶部那个大号价格会被你账号身上挂着的优惠券和当前促销折算，采集当天我这个账号就带着「新用户首月五折」，上面那张CNY截图里Team显示的¥924也是打了60% off的首月价（次月按¥770 /席位续）。**真实标价以页面下方Compare Plans对照表的Price行为准**，上面这张表抄的就是那一行。

核心建议

这个基准价我另外复核过两道，因为全书所有的¥数字都压在它身上。一道是登出复核：**2026-09-13不带账号、不带登录态重取了一次这个页面，Compare Plans的Price行仍然是Pro Monthly \$20.00 / Yearly \$240.00、Max \$90.00 / \$1080.00、Team \$330.00 / \$1980.00，Free那行Exports仍然是15 (H2.5 only)**，和2026-09-11带着优惠券采集的那次逐字一致——券折到的是卡片，没折到对照表。另一道是比值自查：三个付费档的CNY对USD比值全部正好是7.0（月付 $140 \div 20$ 、 $630 \div 90$ 、 $2310 \div 330$ ，年付 $1680 \div 240$ 、 $7560 \div 1080$ 、 $13860 \div 1980$ ），优惠券只会打歪其中一格，不会让这六个比值一起落在同一个汇率上。两道都过了，所以 $\text{¥}140 / 3000 \text{积分} = \text{¥}0.0467$ 这个基准可以接着用。还缺一张无痕窗口下的Pricing页截图做正面证据，我这一轮只复核了页面原文，没拍到图。

几条只在对照表里才看得到、但可能直接决定你能不能用的：

- **Free账号有15次导出，但限定在H2.5这一档：**官方对照表Exports那一行原文写的是 **15 (H2.5 only)**——额度有15次，只是被限定在H2.5这一档模型上。同一张表里Free的Private Models和Commercial Use都是 **✗**，Smart Topology Mesh是Limited Access，所以就算导得出文件，那个文件也不是能放进产品里的。**全书凡是讲到免费档能不能导出，都以这一行为准：**§ 07那句「免费档能试质量、不能交付」就是从它来的。我自己的项目记录里另有一条「GLB / FBX全部导不出、必须订阅」，但我从Pro起步的，那一条没在免费账号上复现过，强度不如这张官方表，两处我都标了出处。
- DCC Bridge（浏览器一键把模型推进Blender / Unity / Unreal等）是Pro及以上才有的。
- 绑骨与动画、Ultra网格质量、多视图转3D，Free都没有。
- 8K贴图Free和Pro是「有限访问」，Max以上才是无限。

核心建议

H2.5是哪一档模型，定价页上没写，但去开发者文档对一遍命名法能对上号。开发者文档（developers.tripo3d.com）把模型分成H系和P系两条线：H系是高精度那条，其中H3对应 v3.0 / v3.1，H2那条标的是「Stable 2.x」、默认 model_version 是 v2.5-20250123；P系是干净拓扑那条，对应 P1.0与P2.0。按这个命名法，对照表里的H2.5就是高精度线上老一代的v2.5，不是工作台默认的v3.1，也不是智能网格那条P线。这一条是我拿开发者文档的命名法推出来的，定价页本身没有明说，所以当参考用，别当官方定义。2026-09-13查。

核心建议

免费档能不能导出，你自己花三分钟就能当场问清楚，而且一分积分不花。三步：①注册后先生成一件（默认配置就行，200积分够）；②点Export，选你项目里真正要用的那个格式，多半是GLB或FBX；③一路点到最后一步，看它是直接给文件，还是弹出升级提示。导出本身不扣积分（见下面第三节最后一行，我拿余额前后读数验过），所以这一步花的是时间，不是那200分。这本书之所以没把这条钉死，是因为我是从Pro起步的，账号上没有免费档可以复现。

同一页切到USD、年付视图（图见 § 07）：Pro标\$20/月、按年计费\$240/年；页面顶部那个「Annually 50% off」是促销横幅，对照表里的年付价并没有再打折。

我用的是哪一档：Professional年付，实付每月约¥91，到期2026-10-08。这是2026-09-11计费接口返回的促销价（标35% off），和官网标价表的口径不一样，**对外一律按官方标价¥140 / 3000积分算**。账户里那两万多积分是另外买的积分包，不是订阅送的。

二、每个操作多少积分（官网FAQ原文）

官方积分表见 § 07那张（tripo3d.com/pricing底部FAQ第一条展开）。那是官方唯一一处把每个功能的积分和可选加项列全的地方。

功能	基础积分	可选附加项
HD Generate（高精度生成）	15	Ultra +15、Generate in Parts +30、Texture Gen +10、HD Texture +10、Quad +5
Smart Mesh（智能网格）	35	Smart Mesh P2.0 +65
Segmentation（智能拆分）	5	Part Completion +5 /每部件
Retopology（重拓扑）	5	Quad +5、Smart Lowpoly +30
Texture Gen（贴图生成）	10	Style Reference +5、PBR +5、HD Texture +10

功能	基础积分	可选附加项
----	------	-------

Auto Rig（自动绑骨）	20	—
----------------	----	---

截至2026-09官方标价，以官网为准。

表里的Smart Mesh 35加P2.0的+65正好是100，但100是界面上被划掉的那个原价，实际扣的是折后的65，见下一节。Segmentation是唯一对不上的一行：表里写5，界面按钮上是40，对不上的地方我按界面记，因为扣钱的是界面。

核心建议

这一行悬着，但它悬不过你的第一次拆件：做一次拆件，前后各读一次顶栏余额，差多少就是多少。余额差法不受界面显示影响，也不用信任何一张表（同一套方法我拿它对过贴图那30分和智能网格那12笔65分，见 § 07「怎么记账」）。两个前提要避免：①别挑挂着Trial的时候拆，Trial实扣0，读不出差额；②拆的那件要是三角面、没绑过骨的模型，四边面和已绑骨的件不在拆件的适用范围里（§ 05）。我这一轮没验，是因为第一条正好撞上了：全书只拆过一次件，就是09-11那辆出租车，那天按钮挂着Trial，实扣0。此后的工作没有一件需要拆件，所以下面第七节那张按余额读数记的流水（09-11下午到09-12）里，从头到尾没有出现一笔拆件扣费——那张表是拿顶栏余额兜住的，它里面没有这一行，就是「我确实没碰到第二次机会」的正面证据。专门为验价再拆一件，等于为一个本轮用不上的结果付一次钱，我没付。在你那边这不用额外花钱：你迟早会拆第一件，拆的时候顺手记两个数就行。

三、界面上实际扣多少

打开生成表单，按钮上直接写着这一单多少积分。我把几何与贴图弹层里的开关一个个拨过去，记下按钮上的数，得到下面这张阶梯表。全程只看按钮，一积分没花。

默认状态（图见 § 01）：Ultra Mesh Quality关、Texture开、Texture Quality 8K、PBR开、Topology Triangle、面数20000，按钮「Generate 50」。

同一张图、同一个表单，只把Texture那一个开关关掉（图见 § 01），按钮变成「Generate 15」——关掉的是一整包东西，不止那一个开关。

表单状态	生成按钮	拆开看
Texture关（只要几何）	15	几何底价
+ Texture，质量2K	30	15 +贴图包15
+ Texture，质量4K	40	15 +贴图包25
+ Texture，质量8K（默认）	50	15 +贴图包35
再开Ultra Mesh Quality	65	50 + 15

截至2026-09界面实测，以当天按钮上的数为准。

其余几项没有档位，一口价：

操作	界面显示	实扣	说明
智能网格（P2.0 Preview）	「100划掉65」	65	原价100被划线打折。出的是无贴图灰模。Preview期首次Trial全免（显示「100划掉0」），之后每笔落在65
智能网格之后加贴图	纹理生成30	30	灰模要能看必须再付这一笔。 一件带贴图的完整成品是65 + 30 = 95
智能拆分	40（当天显示Trial免费）	0（那天）	官网表里写5，界面是40；对四边面模型和已绑骨模型不可用
Retopology + Quad	5 + 5	10	形状本身对了就走它，比重新生成便宜得多
Auto Rig	20	20	碰上第一次没绑成就点重试，那一件是40。不是每次都会碰上，我实测过一次过的
把外部模型上传进Tripo	导入50	50	只有外部文件才需要。从账号内已有资产直接选是0（\$ 06有走法）
导出	不标价	0	导一份FBX，前后两次余额读数一分没变

截至2026-09界面实测，以当天按钮上的数为准。

核心建议

这一节最值钱的一条：界面上那个「生成50」不是几何的价钱，其中35分是整个贴图包。几何本身15，8K贴图包35。贴图档位是你唯一能按用途调的旋钮：主角留8K，中景走4K，远景拨到2K，只要形状的把Texture整个关掉。这和面数那边是同一件事：**默认值替你做了选择**——只不过面数那个默认值恰好好用，贴图的默认是最贵那档，知道了就都能调。

注意

两套加法对不上，我记的是界面这一套。官网FAQ把50拆成HD Generate 15 + Ultra 15 + Texture Gen 10 + HD Texture 10；但界面上默认状态的Ultra Mesh Quality是关着的，把它打开，按钮从50跳到65。两边在「默认一单50」这个结果上一致，在「50由什么构成」上不一致。另外界面把贴图相关的几项收在一个Texture开关底下，要么整包关掉，要么只能降档（8K → 4K → 2K），单独关掉HD Texture这一项是做不到的。

注意

表单会记住你上次的设置。那天早上我看到按钮写着65，第一反应是涨价了。不是，是前一次生成把Ultra Mesh Quality开着，被记住了——弹层顶上那行小字写得清清楚楚：We'll automatically save your settings for next time。开工前先展开弹层核一遍开关，比事后对账省事。**面数那一栏反而不被记住，每件都要重设。**

四、五种配法，按件的用途挑

这件在画面里是什么角色	配法	一件
主角，观众会开到跟前细看	高精度+ 8K贴图（工作台默认）	50
中景，街边的建筑和道具	高精度+ 4K贴图	40
远景，只会远远瞥一眼	高精度+ 2K贴图	30
只要形状，材质在引擎里自己给	高精度，Texture整个关掉	15
要干净四边面，拿去Blender接着改	智能网格P2.0 +贴图生成	65 (+30)

截至2026-09界面实测，以当天按钮上的数为准。

纯几何那一栏才是两种模式的同口径对比：**高精度15，智能网格65**。都带贴图的话是50对95。把50和65并排比是错的，那是一边带8K贴图、一边不带。

五、单次购买积分包

金额	积分	折合
\$10	1000	\$0.010 /积分
\$100	10000 + 2000	\$0.0083 /积分
\$500	50000 + 12000	\$0.0081 /积分
\$1000	100000 + 30000	\$0.0077 /积分

截至2026-09官方标价，以官网为准。订阅自带的积分更便宜：Pro ¥140 / 3000积分= **¥0.0467 /积分**，本书所有折算都按这个数。

六、第一段流水：09-10到09-11

项目	积分	备注
星月夜（含名画可行性）	200	其中50是误判失败的重复提交
四冲程发动机	100	活塞、连杆
会转的游乐场	100	两只木马
星光游乐园（Three.js版）	0	全部复用已有资产

项目	积分	备注
三模式实验E5	40	行人 pedestrian-01 绑骨20 +重绑20
Tripo绑骨实测	70	T-Pose角色生成50 +绑骨20，这一件没有重试
小王子星球	840	16 × 50 +绑骨20 × 2（第一次失败要重试）
五个方案与报告	0	复用
小计	1350	≈ ¥63

注意

这一段的口径，三样东西不在里面：巴黎那条线9月9日之前的生成；Unity那一版（单独记在它自己的文档里）；9月11日下午那批38件的P2.0批量，3640积分，记在下一节。第三样比这张表本身还大，分开记是因为它从头就是拿余额读数记的账，和这张表的记法不是一回事。

还有一件得说在前面：这一段是按每条线的记录加出来的，没有余额读数兜底，所以1350是下限不是总账。回头拿当天零星记下的余额去对，这一段至少有两处对不上：26910掉到26710是200，对得上星月夜；26710掉到26560是150，而发动机那条只记了100；26460掉到26310又是150，没有任何一条记录认领。那几笔多半是当天顺手试开关、重复提交之类的小额操作，我没记，现在也补不出来。这正是我从第二天起改用余额记账的原因。E5那40和绑骨实测那70是两个不同角色的两笔钱，别合并。

七、第二段流水：09-11下午到09-12，六个余额读数

第二段我换了记法，每个节点抄一次顶栏余额，最后拿头尾相减去对逐笔加出来的数。这一段只差10分没对上，下面会把它指出来。

时间	发生了什么	扣	余额
09-11	起点读数	—	25220
09-11	那辆出租车做贴图生成	30	25190
09-11 14:36	批量开工前读数	—	25180
09-11 14:36–15:50	38件智能网格批量（37×65 +1件Trial免费+1件重做65 +38×30 +1次重做30）	3640	21540
09-12开工	第二天开工读数	—	21540
09-12白天	网格14×65=910、纹理13×30=390、绑骨20	1320	20220

时间	发生了什么	扣	余额
09-12	两种模式对照实验E10，v3.1那一件	15	20205
09-12	把结果导出成FBX下载	0	20205（前后不变）
09-12录屏现场	现场生成一件65 + 现场绑骨20	85	20120





两个节点的顶栏余额原始读数：左边20205（E10那一笔15扣完之后，同一张截图里生成按钮写着「100划掉65」），右边20120（当天最后一笔绑骨扣完）。两张之间差85，正好是现场那一笔生成65加一笔绑骨20。Tripo Studio中文界面，2026-09-12。

头尾相减：25220 - 20120 = 5100积分，按¥0.0467算约¥238。逐笔加起来是5090，差10分。

注意

这10分我没核到出处，如实写在这里。缺口在25190到25180这一段：09-11下午做完出租车贴图之后、38件批量开工之前，账上少了10分，而我那一段没记账，多半是在界面上试开关时点出去的某一笔小额操作。跨过这一段之后两头都是干净的：38件那批自己的记录里起始写的就是25180，减3640等于21540，和页面余额一分不差；09-12那一整天也逐笔对得上。如果你只拿25220和21540两个数去减，看起来会缺40，那是因为中间漏掉了出租车贴图那笔30，它是有记录的。真正没主的只有10分。

余额差法的价值就在这儿：它会把你没注意到的那一笔顶出来，而不是让它混进一个漂亮的总数里。一个账本上有个标着「没对上」的缺口，比一个处处严丝合缝的总数可信，因为后者往往是凑出来的。这本书里另一处也是这么处理的：拆件那个功能，官方表写5、界面按钮写40，我至今没拿到一笔能验它的余额差，就一直这么写着。

八、要引用哪个数

对外只说5100那个。它是余额头尾相减出来的，不依赖任何一条记录的完整性，是这本书里最硬的一个成本数字：5100积分，约¥238。

注意

第一段那1350和第二段这5100，不要相加。第一段发生在25220这个读数之前，是另一段账；两段之间没有一个余额读数把它们接上，第一段本身又是下限。硬加出来的那个数看着完整，实际比两个分开写的数更不可信。能闭合的那段单独说，不能闭合的那段标明是下限，这比凑一个总数诚实。

这5100换来的是：一座可驾驶的巴黎、六个能玩的demo、一台能装配的发动机、三个星球、一幅拆得开的画，外加38件加13件两批可复用资产。对个人开发者来说，这个成本结构的含义是：生成可以当草稿手段用，试错的钱远比你以为的少；真正贵的是你决定生成什么，以及生成完之后接进项目的那几个小时。

核心建议

按Pro一个月3000积分算：全走默认8K档是60件，走4K档是75件，走2K档是100件，只要几何是200件。做游戏资产这类需要大量试错的活，先把件的用途分好类再定档，比一刀切省下的不是一点半点。

附录D 术语：用人话讲一遍

Appendix D · Glossary in Plain Language

这本书是写给不会写代码、也没碰过3D的人的，所以每个术语都按「它到底是什么、你什么时候会撞上它」来解释，不抄定义。按你会遇到它们的顺序排。

一、模型本身

术语	人话
网格/ Mesh	3D模型的骨肉。一堆点连成面，面拼成形状。所谓「生成一个3D模型」，生成的就是这堆点和面。
三角面/四边面	面的形状。三角面是最通用的，引擎最后都转成三角形来画；四边面（Quad）是四边形为主的规整网格，做变形、细分、绑骨时更听话，是专业管线的通用标准。Tripo的高精度模型表单里可以选三角面或四边面，默认三角面；智能网格那边默认就是四边面。
三角化/ Triangulate	把四边面和多边面拆成三角形。引擎最后只认三角形，所以这一步一定会发生，区别只在于是你自己做还是导出时自动做。 倍数不是2 ：本书那批38件实测在1.37到1.87倍之间，中位1.75，因为生成出来的网格里总混着本来就是三角的面。做面数预算时用1.75去估比用2稳。
face_limit（面数上限）	生成表单里让你填的那个数。有两个地方会咬人： 它数的是什么，随模式变 ：智能网格那边数的是四边面，高精度模型那边数的是三角面，同一个8000在两边不是一回事； 它是目标不是硬上限 ，38件实测有35件超出设定值，中位超13%，最狠的一件超了44%。要卡死预算就往下压一档填。
拓扑	这些面是怎么排布、怎么连接的。同样一个形状，面排得整齐叫「拓扑好」，乱七八糟叫「拓扑差」。形状对不对看眼睛，拓扑好不好要看线框。
面数预算/ Polygon Budget	目标设备能扛住多少三角面。手机上跑的和电脑上跑的不是一个量级，超了就掉帧。它是一个你自己定的上限，不是系统给的。
减面/简化	把面数砍下来，形状尽量不变。能砍到多少由几何的光滑程度决定，光滑的（人物）砍得狠，棱角多的（铁塔）砍不动。
重拓扑/ Retopology	不是砍面，是 重新铺一张干净的网 。生成出来的模型面排得乱，重拓扑之后线走得规整，再去绑骨、做动画、做细分才不出问题。Tripo上5积分，加四边面再5积分，比重新生成一次（50）便宜得多。

术语	人话
包围盒/ Bounding Box	刚好把模型装进去的那个长方体。量尺寸、对齐、算缩放全靠它。⚠️对带骨架的模型，轴对齐包围盒会骗人（见附录B）。
UV与贴图	贴图是一张平面的图，UV是「这张图怎么裹到模型表面上」的对应关系。颜色、花纹、锈迹全在贴图里，不在几何里。
PBR	基于物理的渲染。模型不只带一张颜色图，还带金属度、粗糙度、法线等几张图，引擎按真实世界的光照规律算出效果。它是「为什么你的模型在白底上渲成一团黑」的根源，金属度高的材质要靠反射环境才有亮度。
基色贴图/ base color	纯颜色那一张，不含任何光照信息。你看到的「这栋楼是奶油色的」就来自它。智能网格（P2.0）模式只给这一张，没有法线也没有ORM，所以同一盏灯下它会比三张齐全的那种哑一点。
ORM贴图	一张图管三件事，靠三个颜色通道分工：R存环境光遮蔽、G存粗糙度、B存金属度。合成一张是为了省显存和采样次数。v3.1（高精度模型）导出的三张贴图里就有它，P2.0那边没有。名字里的三个字母就是这三个通道的英文首字母。
法线贴图	一张假装凹凸的图。它不改变几何，只骗光照，让平的表面看起来有纹理。它的分辨率可以比颜色图低很多。
绑骨/ Rigging	给模型装一副骨架，并且告诉每一块表皮「你跟着哪根骨头动、跟多紧」（后半句叫蒙皮权重）。装完之后模型才能被动画驱动。Tripo的自动绑骨20积分，出来41个关节。
动画重定向/ Retarget	把一套现成的动作（比如走路）套到另一副骨架上。Tripo动画库里那近一百个动作就是这么用上的，它单独出一个文件，和模型文件是分开的。
LOD	同一个模型准备几个精度版本，离得远用粗的，近了才换细的。巴黎那个城市demo全靠它撑着：近处165米内用Tripo生成的整棵树，再远换成程序化的树冠，更远只剩树干。它出事的方式不是掉帧，是东西凭空消失，见附录B。
实例化池与配额	同一个模型要画几百份（树、路灯、长椅）时，不是画几百次，是把所有位置塞进一个「池子」一次画完，这叫实例化（InstancedMesh）。池子有容量上限，那个上限就是配额。配额满了之后怎么办，比配额定多大更要紧：退而画简化版是对的，什么都不画就会让近处的东西凭空消失。实测把配额从36调到110，三角面从144万涨到244万，绘制调用一个没增。
glTF / GLB	3D模型的通用交换格式，GLB是把模型、贴图、动画打包成一个文件的版本。Tripo导出、Three.js加载、Unity导入，走的都是它。

术语	人话
meshopt	一种给GLB瘦身的压缩扩展（EXT_meshopt_compression）。Tripo的CDN上给的就是压过的版本，好处是下载快，坏处是 有些工具解不开 ，会表现成「文件明明在，就是加载不了」。本地跑一次 <code>glTF-transform copy</code> 就能解开。
KHR_mesh_quantization	另一种给GLB瘦身的扩展，做法是把顶点坐标从32位浮点压成16位整数。它和meshopt的关键区别是 Three.js原生就认，不用外挂解码器 ，所以遇到它不用慌。唯一的坑是：压的时候会标一个「归一化」标志，不认这个标志的工具会把整数当成真实坐标读，于是一个1米高的模型被读成32767米。 看到32767或它的倍数，八成就是这件事。
扫掠回退/ sweep	碰撞检测的一种做法：不只看「这一帧我在哪、有没有插进墙里」，而是把上一帧到这一帧之间那一整段路扫一遍。速度快的时候，只看单帧位置会直接穿墙过去，扫掠能兜住。代价是每帧多算一点。

二、Tripo平台上的词

术语	人话
积分/ Credits	平台的计费单位。每次生成和编辑按复杂度和设置扣，套餐按月给，也可以单独买。Pro是¥140换3000积分。
图生3D / 文生3D	喂一张图出模型，还是喂一段文字出模型。本书全程走图生3D，因为 你对结果的控制力几乎全在那张输入图上 。
HD Model（高精度模型）	Tripo的主力生成模式，我绝大多数资产都出自它（v3.1最高质量）。界面上标50积分，那是默认替你开好了整个贴图包的价，只要几何的话15就够。
Smart Mesh（智能网格）	另一种生成模式，直接出规整的低面数网格（P2.0预览版实测一辆出租车5615个四边面），但出来是没有贴图的灰模，贴图要另付30积分。 而且它的贴图只给一张基色 ，没有法线、没有ORM，接进场景前要知道这件事。
语义分件/ Part Segmentation	把一个整体模型按结构自动拆成可以单独选中、单独操作的部件，比如车轮从车身里分出来。它和「按连通分量拆」完全不同，后者会把一个模型炸成几百个碎片。⚠️ 四边面模型和已绑骨模型不能做这一步。
Trial x1	新功能的一次性免费试用，界面上会把价格划掉显示0。 做预算时别把试用价当常态 ，但也别把免完之后的价记成标价上限——Preview期的智能网格免完之后落回的是划线折扣价（100划掉、实扣65），不是回到100。而且它只免生成那一笔，贴图照付。
DCC	Digital Content Creation，Blender、Maya、3ds Max这类专业内容创作软件的统称。

术语	人话
DCC Bridge	Tripo提供的一条传输通道：浏览器里生成完，一键把模型推进Blender / Unity / Unreal等软件，不用手动下载再导入。它和AI agent无关，只是搬运，Pro及以上才有。

Production-Ready	生产就绪，意思是生成结果不用大幅返工就能进专业制作管线。判断它成不成立，看的是拓扑、UV、尺寸这些，不是看渲染图好不好看。
------------------	---

三、把东西接起来的词

术语	人话
Vibe Coding	用自然语言说清你要什么，由编程Agent完成编码与调试的开发方式。本书所有demo的代码都是这么写出来的，我一行代码都没有手写。
GPT-6 / Astra	OpenAI在2026年9月发布的新一代模型。本书用到它的地方只有一处：它驱动的编程Agent（Codex）替我写完了所有demo的代码，以及生成本书里的设计图。书里没有测过它操作Blender之类专业软件的能力，别把我没做过的事算到它头上。
Codex	本书里指我用的那个编程Agent。它在这本书里干两件事：写代码，以及出设计图。
MCP	Model Context Protocol，一套让大模型连接并操作外部工具的协议。装上对应的MCP server，Agent就能直接调Tripo的生成能力，或者直接指挥Blender。
Three.js	在网页里画3D的库。你做出来的东西发个链接别人就能玩，这是它相对引擎最大的好处。
环境贴图/HDR	一张包裹整个场景的全景图，用来告诉模型「你周围有什么光」。没有它，PBR的金属材质就没有东西可反射。它不一定要是外部HDR文件，自己用canvas画一张渐变就够用（代码在附录B）。
glTFast	Unity里加载glTF / GLB的包。两个你一定会撞上的脾气：默认不带meshopt解码器（装com.unity.meshopt.decompress 就认，不是解不开），以及绕过Unity的贴图导入设置（所以平台贴图压缩不生效，WebGL包会大得离谱）。
WebGL构建	把Unity项目导出成能在浏览器里跑的一份网页。体积是它的主要矛盾，我实测从165 MB压到51 MB，靠的全是贴图。
Toy	B站的互动内容平台，可以把一个前端项目或HTML发布上去、绑定到视频下面，观众看完视频直接点进去玩。它是我这类demo最顺手的分发出口之一。

术语	人话
算子耗时 / 端到端 墙钟	看到一个耗时数字，先问它是哪一个。 算子耗时 是平台报的、模型真正在算的那一段； 端到端墙钟 是你坐在屏幕前从点下按钮到能下载为止真实等了多久，中间还有上传、排队、生成预览、下载。录制那天现场实测：生成端到端约31秒，其中算子16秒；自动绑骨端到端13秒，其中算子3秒（另一次算子19秒）。本书凡是引用平台侧的秒数，一律是算子耗时口径，不含上传与排队。只有一处例外，看到别当矛盾： 高精度模型那一档我没有逐件采到算子耗时 ，书里给的「一件约四分钟」（\$ 05、\$ 07那张耗时表）是端到端墙钟，口径说明在\$ 05。所以「高精度模型一件约四分钟」和「生成算子16秒」摆在一起不是打架，是两个口径，别拿来互相换算。
无头浏览器/ headless	没有界面的浏览器，用命令行驱动它打开页面、截图。做自动化自检必须用它，因为放在后台标签页里的页面会被浏览器暂停，你截到的是旧画面。

核心建议

这些词里，真正需要你记住的只有三个：**拓扑**（决定模型好不好用）、**PBR**（决定它渲出来好不好看）、**积分**（决定你敢试几次）。其余的等你撞上了再回来查。

3D Vibe Coding手册

从一句话到能跑的场景：GPT-6 Astra × Tripo全流程实测

合上书之后，下一步在这儿

注册就有200积分，官方自己标的是「约13个模型」——够你把 \$ 01那一趟从头走一遍，先看看这条路合不合手。积分怎么花在刀刃上，\$ 07有一整节的账。

开始做：studio.tripo3d.com

定价与积分：tripo3d.com/pricing · 开发者文档：developers.tripo3d.com



微信搜一搜

🔍 花叔

花叔 · AI Native Coder

我一行代码都不会写，却用AI做出了AppStore付费榜Top 1的小猫补光灯，写了十二本免费的技术手册，这本是其中最厚的一本。

所有产品，全部AI写的，我只负责想清楚要做什么。

开源了女娲.skill、huashu-design等项目。

我始终相信：AI生成内容不稀缺，稀缺的是判断力。

B站：[花叔v](#) · 公众号：[花叔](#) · [X/Twitter](#) · [YouTube](#) · [GitHub](#) · [官网](#)

Created by花叔 · v260913 · 2026年9月

这本书是B站那条「用AI把一张图做成一座能开车的巴黎」的随片资料。所有橙皮书免费下载：huasheng.ai/orange-books