arXiv:2301.06702v1 [cs.CR] 17 Jan 2023

# *Shackled*：一款完全以以太坊智能合约编写的三维渲染引擎

Ike Smith[1] 和 Casey Clifton[1,2]

[1]Spectra，澳大利亚 www.spectra.art
ike@spectra.art，caseyclifton@proton.me

**摘要**。 以太坊区块链允许开发和部署智能合约，这些合约可以存储并执行“链上”代码——也就是说，完全由区块链网络中的节点执行。智能合约传统上用于金融领域，但由于智能合约是图灵完备的，其算法范围远超任何单一领域。为此，我们设计、开发并部署了一款名为 Shackled 的综合三维渲染引擎，完全以以太坊智能合约编写。Shackled 可从三维场景计算出二维图像，所有计算均在以太坊上链上执行。据我们所知，Shackled 是首个也是唯一一个完全在以太坊上运行的链上三维渲染引擎。在本文中，我们 1）提供三个独特的数据集，用于使用 Shackled 和对其进行基准测试；2）执行上述基准测试并提供结果；3）展示 Shackled 在代币化生成艺术领域的一种潜在用例；4）为 Shackled 提供无代码用户界面；5）列举使用 Solidity 智能合约编写复杂算法所面临的挑战；6）概述改进 Shackled 平台的潜在方向。我们希望这项工作能够提升以太坊区块链原生的图形处理能力，并推动智能合约用于更复杂的算法，从而丰富整个以太坊生态系统。

**关键词：** 三维渲染 · 渲染引擎 · Blinn-Phong 光照 · 链上 · 以太坊 · 区块链 · 智能合约 · Solidity · 非同质化代币 · 非同质化代币

## 1 引言

以太坊区块链常被称为“世界计算机” [B+]，因为它提供了一个智能合约平台，允许通用算法“链上”运行——也就是说，完全在区块链网络的节点上运行。智能合约代码具有一些有趣且新颖的优势 [ZXD+20]，具体包括：不可变性（记录一旦写入区块，就无法在以太坊上修改）、永久性（只要区块链持续存在，代码就会一直存在）、透明性（所有

[2] 通讯作者。

智能合约源代码都公开可见）、可扩展性（智能合约代码可在网络中的任何节点上执行）、精确的可复现性（其他程序在给定相同输入时，可以完全一致地执行该代码），以及可组合性（其他程序可以将这段代码用作更大系统中的构建模块）。

在这项工作中，我们完全使用 Solidity 智能合约编写了一个三维渲染引擎——也就是说，一个以三维场景为输入，并计算生成一幅二维图像作为输出、呈现该场景从给定相机姿态看上去的样子的程序。随后，我们将其部署在以太坊区块链上。由于该引擎具有链上特性，我们将其命名为 $Shackled$ 。据我们所知，**Shackled 是首个完全链上的三维渲染引擎**。

这项工作的贡献如下：

1. 对 Shackled 的效率进行量化，以 Gas 用量为衡量指标。2. 与 Shackled 兼容的三个三维对象数据集。3. 展示 Shackled 的实用性：我们开发并部署了一个生成艺术项目（$Shackled\ Genesis$），将 Shackled 与非同质化代币（NFT）集成。4. 为 Shackled 智能合约库提供一个名为 ***Shackled*** 创作**工具**[1], 的用户界面，让用户通过网络上的简单、无需编写代码的可视化界面，完全在区块链上渲染三维物体。5. 全面讨论将复杂算法编写为 Solidity 智能合约所面临的挑战。

## 2 相关工作

**Solidity 智能合约** 是一项相对年轻的技术 [Dan17], ，主要用于金融应用场景。最值得一提的或许是，它推动了去中心化金融（DeFi）的发展；这是一类以基于区块链的去信任化和无需可信第三方即可进行安全点对点交易为特征的金融 [CPP21]。然而，智能合约的应用远不止金融领域，以太坊生态系统中已出现复杂算法的链上实现（例如链上国际象棋引擎 [fiv21]）。

智能合约一项新颖却颇具争议的应用，是对艺术进行**代币化** 并发行非同质化代币（NFT） [WLWC21,Cho21]。非同质化代币因其对传统版权法的影响 [Gol21], 、侵犯知识产权[Yod22], 、对艺术的影响 [Fry21], 、环境影响 [TBDI22] （本项目在附录A的相关背景中对此进行了讨论）、在现代时代思潮中的过度炒作 [MB21], ，以及缺乏监管助长了诈骗、操纵和欺诈等掠夺性文化 [TM20]而受到批评。

相反，它们也因重新定义了数字时代的所有权性质而受到赞誉 [WLWC21,Cho21], ，并被视为一种新颖的投资机会，

[1] 可通过 https://shackled.spectra.art/#/creator 访问。

以及它们对艺术文化 [Kug21] 和收藏行为所产生的积极 影响。

关于艺术代币化，已有大量研究和实践，包括基于以太坊的生成艺术项目 - [Hub21、HA21、Nou21], 链上图形库[Wu21], 将三维物体存储在链上 [say 22], 以及利用链上技术生成和保存数字艺术 [MJ19、MJ17、Hol22]。

艺术代币化与智能合约编程相交之处，蕴藏着在链上执行三**维渲染** 的潜力。目前，人们正在探索并商业化去中心化三维渲染方案 [Tok17]，这些方案着重于在定制区块链上实现大规模计算，并利用计算节点图 [Alc22、WJL+20] 进行图形处理优化。这些方案将硬件负担从用户转移到去中心化的计算服务提供者网络，实质上为专用计算创造了一个市场。

尽管 Solidity 智能合约为算法开发带来了巨大机遇，但目前其支持的计算规模有限；现代图形引擎目前还无法用 Solidity 实现。因此，Shackled 基于早期图形先驱的研究成果，采用了近 50 年前的技术（这些技术更适合当今在链上实现）。具体而言，我们修改了 Bui Tong Phuong [Pho75]和 Jim Blinn [ Bli77] 提出的原始三维渲染与光照模型，并以此创建了一个 Solidity 版本的简单渲染管线，其设计灵感来自 OpenGL[SA99]。

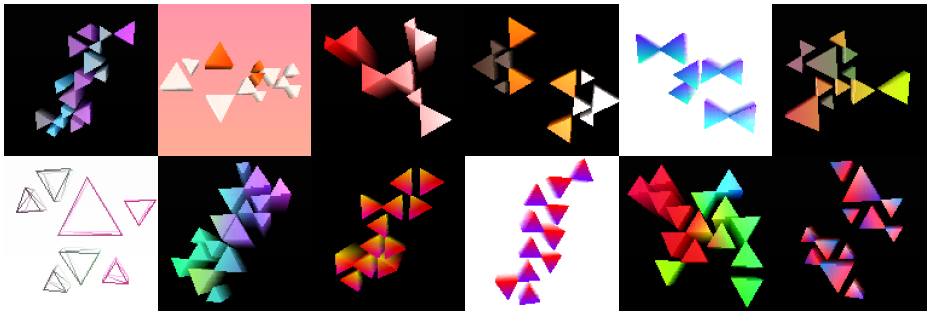
## 3 数据集

我们首先开发并在表 1 中介绍了三个供 Shackled 使用的数据集，这些数据集将在本文中贯穿使用（数据可用性见附录 B）。首先是**Shackled Genesis 数据集**，其中包含 1024 个通过链上几何体生成算法生成的基于三棱柱的几何体。其次是**Shackled Icons 数据集**，由一系列手工制作的三维物体组成。第三是**Shackled Common Graphics Objects Dataset**，由三维图形学史上的常见物体组成，并经过处理以兼容 Shackled。

## 4 方法

### 4.1 开发

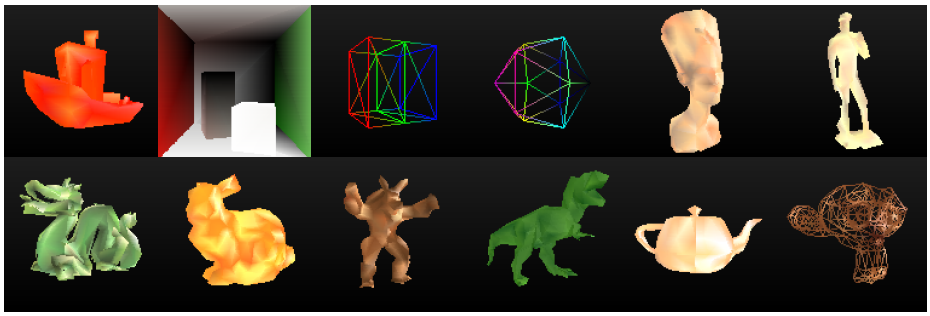
Shackled 使用以太坊上编写和测试 Solidity 智能合约的常用开发工具开发。我们的**技术栈**包括作为主要开发环境的$Hardhat$ 、用于与智能合约交互的$Ethers$ ，以及用于测试的$Chai$ 。我们在表 1 中汇总了开发过程中遇到的挑战及其解决方法。



(a) Shackled Genesis 数据集中的示例，渲染效果如图所示。每个实例都包含以生成方式摆放的若干三棱柱，并使用不同的颜色和光照参数进行渲染。该数据集共有 1024 个实例。




(b) Shackled Icons 数据集中的示例，渲染效果如图所示。选择每个渲染对象时，考虑了其辨识度以及与链上艺术和非同质化代币社区的相关性。



(c) Shackled 通用图形对象数据集中的示例，渲染效果如图所示。**上排**：3DBenchy（3D 打印基准模型）、Cornell 箱 [GTGB84],立方体和等角球体的线框渲染图、奈费尔提蒂胸像的 3D 扫描 [WB82], , 以及米开朗基罗《大卫》雕像的 3D 扫描 [LPC+00]。**下排**：斯坦福龙、兔子、犰狳和 Tyra [Sta14], 犹他茶壶，以及 Blender 的 Suzanne [Bla19]。

图 1：本研究使用的数据集。

表 1：Shackled 开发过程中遇到的挑战及克服这些挑战所采用的解决方案。我们将这些挑战归类为由 Solidity 或以太坊生态系统所带来的问题。

| 描述 | 解决方案 |
|---|---|
| *Solidity programming language challenges* | |
| 不支持浮点数或定点数。 | 以 Gwei 为单位时使用整数（$10^9$）。 |
| 将小于 32 字节的数据类型合并到同一存储槽中，从而降低除法精度。 | 填充变量，确保它们占用 32 字节，并使用单元测试确保行为正确。 |
| 溢出和下溢错误很常见，因为整数的大小是固定的。 | 最新版本的 Solidity 包含自动检查，可检测这些错误。 |
| 最佳实践文档有限。 | 从示范项目中学习（所有链上代码都是开源的）。 |
| 堆栈跟踪错误信息缺乏参考价值。 | 使用诸如 Hardhat. |
| 类型转换受限或无法进行。 | 为无法转换为其他类型的类型开发变通方案。对此问题未必有一种“万能”解决方案。 |
| 由于堆栈深度限制，同时只能使用 16 个变量由于堆栈深度限制。 | 使用结构体存储和传递变量。 |
| *Ethereum ecosystem challenges* | |
| 超出节点施加的 Gas 限制服务提供商。 | 运行自己的节点，或寻找愿意提供更高限制的服务提供商（本例中为 Alchemy）。 |
| 修改／写入存储会产生高昂的以太坊区块链上的费用。 | 没有通用的解决方案，但链上压缩和生成可用于避免直接存储数据。 |
| 真正的随机性并非 可能的方式是以太坊是确定性的。 | 伪随机性可以通过使用区块哈希或编号作为输入来实现。 |
| 以太坊虚拟机（EVM）堆栈只有 1024 个槽位可供函数调用其他函数时使用。 | 使用循环代替递归。 |

为了对 Shackled 进行基准测试并**估算计算成本**，我们估算**Gas**的用量；Gas 是一种计量单位，用于描述在以太坊网络上执行特定计算所需的计算能力 [B+]。我们的 Gas 估算方法是对 Gas 限制的上限和下限进行二分搜索，直到触发“Gas 不足”错误；我们不断尝试使用越来越少的 Gas 完成渲染，直到收敛到执行特定渲染操作所需的真实 Gas 用量，从而准确测量完成给定渲染所需的计算成本。

**重要的是，Shackled 执行渲染操作无需消耗 Gas**。整个渲染操作都在只读调用中实现，因此不会向以太坊区块链写入任何数据。**因此，本研究中给出的所有 Gas 估算均为**等效 ***Gas***，例如，Shackled 的渲染基准测试结果为 100 亿单位 Gas

并不是真正消耗了那么多 Gas，而是其计算成本相当于执行一项会消耗该数量 Gas 的操作。

### 4.2 渲染管线的设计

Shackled 使用受 OpenGL 启发的顺序渲染管线，将三维模型转换为二维图像 [SA99]，并针对链上实现与执行进行了修改（见图 2）。主要步骤如下。

**顶点规范**包括提供物体网格中所有顶点的三维位置和颜色。按照 `.obj` 文件格式规范 [MB08]，使用这些三维向量构建面。

**顶点着色器** 传统上是用一种专为图形硬件加速计算而设计的语言编写的程序，可并行处理彼此独立的顶点计算。在 Shackled 中，我们不具备这种加速能力。顶点着色器仍承担传统职责：使用透视或正交摄像机矩阵投影各点，并通过模型视图矩阵变换各点（支持平移和缩放），将各点转换到世界空间。

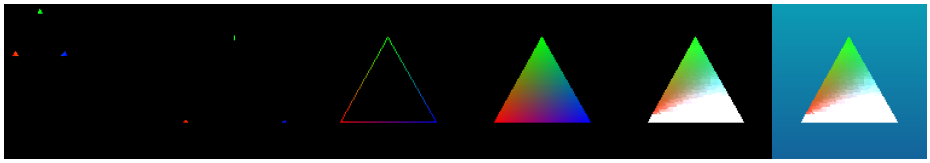


图 2：Shackled 渲染管线对单个三角形进行渲染时各关键步骤的结果。**从左到右**展示的输出依次为：1）顶点规范，2）顶点着色器，3）图元装配，4）光栅化，5）片段着色器，6）合成并返回图像缓冲区。最终渲染结果为一个红、绿、蓝三色的三角形，带有从右下方照射、衰减迅速且强烈的镜面光，并合成在蓝色渐变背景上。所有步骤均完全在智能合约中、以太坊区块链上执行。

**图元装配** 是指将每组三个顶点组合起来，构建成三角形（Shackled 仅支持以三角剖分对象作为输入）。三角形的顶点索引在顶点规范阶段指定。此阶段不执行任何形式的裁剪或视锥剔除。

**光栅化** 结合布雷森汉姆算法 [Koo87] 和扫描线算法 [LC79]，将一组三个顶点转换为线框三角形，再转换为填充三角形，并对颜色进行插值。未实现任何形式的抗锯齿或亚像素插值。在此阶段，片元（候选像素）可能会重叠（即在最终图像中具有相同的 (x,y) 位置），但如果超出画布边界，就会被丢弃。

**片段着色器** 根据提供的光照配置，对每个片元应用布林-冯氏着色 [Pho75,Bli77] 。每个片元的投影深度也用于深度测试，只保留距离最近的片元（即，如果物体的近处部分和远处部分占据渲染结果中的同一位置，则渲染近处部分）。

**合成**是将像素数据写入图像缓冲区（即一个二维矩阵，其中每个元素都是一个 RGB 元组），并将其叠加在背景上的过程。Shackled 支持生成单色背景或双色垂直渐变背景。随后，图像缓冲区会被编码为位图图像。

### 4.3 部署

Shackled 的链上代码由 13 个智能合约和库组成。部署 Shackled 花费了 $0.258\Xi$（当时约合 427 美元），部署 Shackled Icons（后续推出的代币化艺术收藏）花费了 $0.05\Xi$（当时约合 80 美元）。后续项目的成本大幅降低，因为渲染引擎代码已经部署完成，并且可以通过可组合的方式使用。已部署并经验证的合约详见附录 B。这些项目将 Shackled 的渲染结果铸造成 ERC-721A 代币（即非同质化代币），展示了 Shackled 在小众渲染应用场景中的商业潜力和可行性。

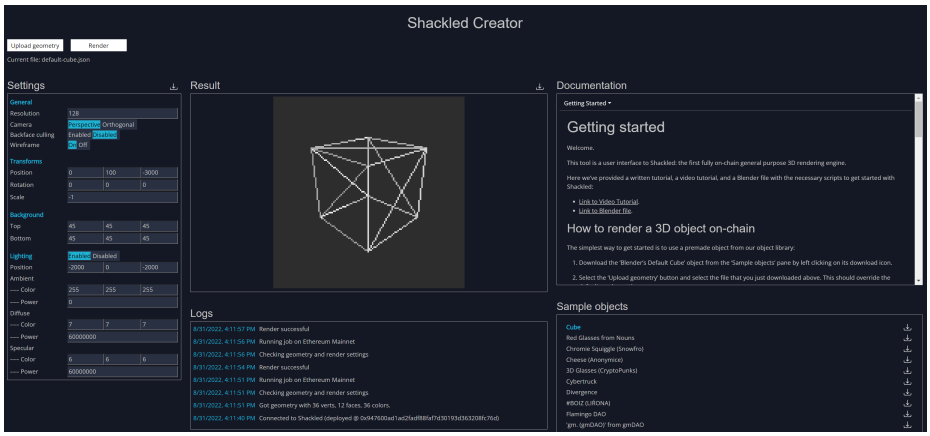



图 3：Shackled 创作者；一个用于 Shackled 渲染引擎合约的简单用户界面。请注意，旋转以变换的形式提供，但旋转矩阵乘法是在前端离链计算的。所有其他计算均在链上执行。

我们还为 Shackled 智能合约实现、部署并提供了一个定制**用户界面**（UI），让用户无需具备链上编程、Solidity 或智能合约方面的专业知识，也能使用 Shackled 进行链上

三维渲染。这款 UI——Shackled 创作者 ——免费使用，并向公众开放[2]。

## 5 结果与讨论

### 5.1 计算量随画布尺寸呈二次方增长

我们使用画布尺寸从 $8 \times 8$ 到 $128 \times 128$ 的画布渲染同一物体，每一步都将两个轴上的画布尺寸加倍。我们使用 Gas 估算流程估算计算成本，并将结果绘制在图 4（右）中。我们计算了两组结果：一组使用透视相机投影模型，另一组使用正交相机投影模型。相应的渲染结果见图 4（左）。

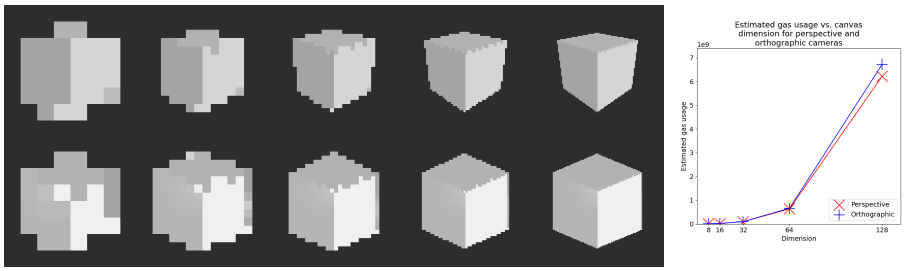



图 4：在链上渲染不同的立方体，同时改变画布尺寸和相机模型参数。**左侧，上排**：使用透视摄像机，将画布尺寸从 8 像素调整至 128 像素。**左侧，下排**：使用正交摄像机，将画布尺寸从 8 像素调整至 128 像素。**右侧**：绘制使用透视和正交摄像机设置进行每次渲染所得的燃料估算值与画布尺寸的关系。

我们注意到，计算成本随画布尺寸呈二次方增长，这符合预期，因为片元数量也会随画布尺寸呈二次方增长。此外，计算成本不受相机投影模型影响。这种二次方增长极大地限制了 Shackled 对更大规模渲染的可扩展性；第 6 节提出了可能有助于缓解这一问题的改进方案。

[2] 可通过 https://shackled.spectra.art/#/creator 访问。

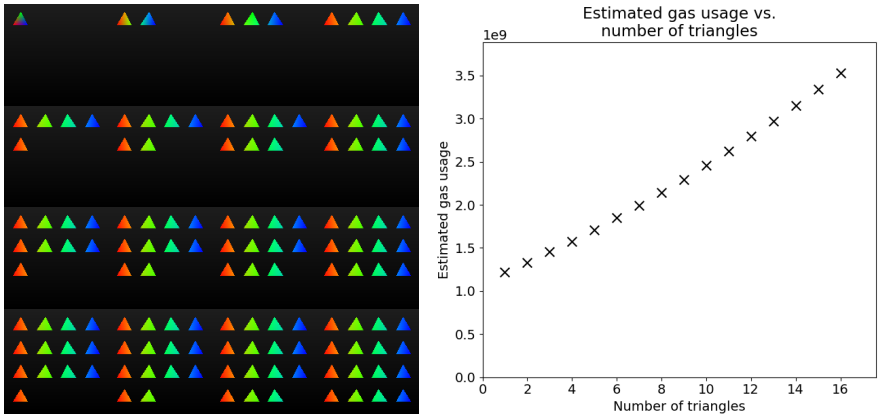



图 5：**左**：测试 Gas 成本如何随渲染像素数量变化所得到的渲染结果。通过渲染 1—16 个相同的三角形来研究 Gas 与像素的关系；最终渲染中每个三角形产生的像素数量几乎相同。**右**：估算的 Gas 用量随三角形数量（因而也随像素数量）近似线性增长。我们注意到，y 轴的偏移表明，无论渲染内容如何，始终都会“支付”固定数量的 Gas。

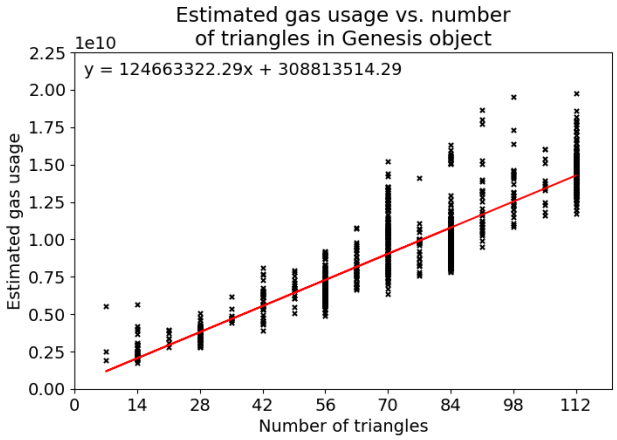



图 6：针对 Shackled Genesis 数据集（见表 1）中的每个实例，绘制估算的 Gas 成本与渲染物体中三角形数量的关系图。使用 NumPy 包的`polyfit` 函数 [HMvdW+20] 进行线性拟合，并绘制和显示结果。每个实例中的每个三棱柱由 7 个三角形组成：前面 1 个，三棱柱的 3 个侧面各有 2 个（三棱柱没有背面），因此 x 轴上的间隔为 7 个单位。

### 5.2 计算量随渲染三角形数量线性增长

我们通过逐步渲染同一几何体（一个三角形）的多个副本，研究渲染数量不断增加的像素所需的成本，如图5所示。我们注意到，三角形数量（因此片元/像素数量）线性增加时，计算成本也会线性增加（以估算的 Gas 近似）。

只要屏幕上的像素数量保持有界，Shackled 就能很好地扩展到包含大量三角形的几何体。这使 Shackled 能够在链上成功渲染种类丰富的物体，如表 1 所示；这些物体中的三角形数量从 20 个（默认立方体）到 1198 个（米开朗基罗的大卫像）不等。

此外，我们使用完整的 Shackled Genesis 数据集（样本见表 1）展示了类似的规律。结果如图6所示，再次表明渲染像素数量与计算成本（同样以估算的 Gas 近似）之间大致呈线性关系。

### 5.3 背面剔除显著减少计算量

此外，我们研究了背面剔除 [Bli93] ——一种通过分析三角形的法向量与摄像机朝向之间的关系，剔除最终渲染中永远不可见的三角形，从而大幅减少所需计算片元数量的方法。通过减少片元数量，我们降低了所需的计算量，从而提升了渲染引擎的能力。

我们分别启用和不启用背面剔除，对 Shackled Common Graphics Objects 数据集进行渲染。不启用背面剔除时，渲染数据集中任意物体的平均 Gas 估算值为 $1.54 \times 10^{11}$；启用背面剔除后则为 $4.39 \times 10^{10}$，减少了 3.51 倍。由此我们得出结论：实现链上背面剔除能够大幅提升 Shackled 渲染复杂物体的能力，同时不会对渲染结果造成任何可见变化。

## 6 结论与未来工作

在本研究中，我们介绍并实现了 Shackled，这是首个完全链上的三维渲染引擎。我们使用三个定制数据集对其效率进行了基准测试，并展示了它在去中心化渲染、代币化艺术，以及作为用于原生三维图形处理的基于区块链的平台方面的应用场景。Shackled 的数据集、源代码和用户界面已公开提供（见附录 B）。此外，我们概述了完全使用 Solidity 智能合约实现复杂算法（例如线性代数方程、三角学和张量运算）所面临的挑战。

在这一早期阶段，Shackled 仅展现了完全链上算法——尤其是渲染引擎——潜力的一小部分。可以改进 Shackled 的**未来工作**方向包括：

– 通过采用数字微分分析算法而非 Bresenham 算法[Koo87], ，消除直线和边缘上的渲染伪影，因为该算法更适合仅使用整数运算进行插值。– 使用四元数、罗德里格斯旋转、矩阵乘积或其他合适的三维旋转算法，实现链上旋转变换。– 充分利用三维图形渲染「易于并行」的特性，例如同时在多个节点上渲染同一渲染结果的不同区块，并在完成后将它们合成。

## 致谢

我们想感谢*Spectra* 的整个团队 在幕后付出的努力，感谢他们分享了大幅改善 Shackled 最终渲染效果的想法，也感谢他们支持 Shackled 及其相关项目的开发。

此外，运行 Shackled 会消耗相当于较多 Gas 的计算量（与传统以太坊智能合约函数相比）。我们想感谢合作伙伴*Alchemy* 特别提高其节点上以太坊读取调用的计算限制，使 Shackled 得以运行。

最后，如果没有*Spectra* 社区，Shackled 的创作便不可能实现。我们衷心感谢 Doubtingtrev、HollywoodMeta.eth、Awesomerrificus 和 Max Bridgland，他们直接参与社区管理、推广、支持与版主工作。

此外，我们还要感谢 Aubjectivity、nodallydude、Bon(g/j)e、Paul Balaji、Ntando Mhlungu、Divirzion.eth、a7111a.eth、I2DT、wakeupremem- ber.eth、Ian Orz、Animechanic、TriPoloski、ZiK WaN、NFTxDeFi、_Michael Slonim ，cearwylm、PetterRasmussen、rpl.eth、Dalst、reechard.eth、JP、El Citadel、Ralph Clayton、Parker Thompson、Ott Erlord 和 Jordysure 对 Shackled 的支持、反馈与认可。若没有你们，便不可能营造出一个积极的社区，让大家为链上艺术的技术进步而振奋。

## 参考资料

Alc22. Alchemy. Alchemy 节点提供商文档。2022。访问日期：2022年8月9日。网址：https://docs.alchemy.com/docs/。

B+. 维塔利克·布特林等人。以太坊白皮书。

Bla19. John M Blain. *The Complete guide to Blender graphics: computer modeling & animation* 2019 。AK Peters/CRC Press, 。

Bli77. James F Blinn. 《计算机合成图像的光反射模型》。载于 *Proceedings of the 4th annual conference on Computer graphics and interactive techniques* 192 198, 1977 ，页码 – 。

Bli93. James F Blinn。背面剔除问题（渲染算法）。 *IEEE Computer Graphics and Applications* 13 6 94 97, 1993 ，（）：–。

Cho21. Usman W Chohan。非同质化代币：区块链、稀缺性与价值。 *Critical Blockchain Research Initiative (CBRI) Working Papers*，2021。CPP21. Francesco Maria De Collibus、Alberto Partida 和 Matija Pi˘skorec。智能合约在四种关键的、基于以太坊的去中心化金融抵押品代币交易网络中的作用。载于 *International Conference on Complex Networks and Their Applications* 792 804 2021 ，页码 – 。施普林格，。Dan17. Chris Dannen。 *Introducing Ethereum and solidity*，第1卷。施普林格，2017。fiv21. fiveoutofnine。链上国际象棋项目。2021。网址：https://www.fiveoutofnine.com/。Fry21. Brian L Frye。非同质化代币与艺术的消亡。 *Available at SSRN 3829399*，2021。Gol21. Matt Goldman。非同质化代币：区块链技术狂野西部中的版权影响。2021。GTGB84. Cindy M Goral、Kenneth E Torrance、Donald P Greenberg 和 Bennett Battaile。漫反射表面间光线交互的建模。 *ACM SIGGRAPH computer graphics* 18 3 213 222, 1984 ，（）：–。

HA21. Harri 和 Arran。Brotchain。*Divergence*，2021。网址：https://brotchain.art/。HMvdW+20. Charles R. Harris、K. Jarrod Millman、St´efan J. van der Walt、Ralf Gommers、Pauli Virtanen、David Cournapeau、Eric Wieser、Julian Tay-lor、Sebastian Berg、Nathaniel J. Smith、Robert Kern、Matti Picus、Stephan Hoyer、Marten H. van Kerkwijk、Matthew Brett、Allan Haldane、Jaime Fern´andez del R´ıo、Mark Wiebe、Pearu Peterson、Pierre G´erard-Marchant、Kevin Sheppard、Tyler Reddy、Warren Weckesser、Hameer Abbasi、Christoph Gohlke 和 Travis E. Oliphant。使用 NumPy 进行数组编程。 *Nature*，585(7825)：357–362，2020年9月。网址：https://doi.org/10.1038/s41586-020-2649-2。Hol22. PROOF Holdings。Moonbirds，官方 PROOF 个人资料图片。2022。网址：https://www.moonbirds.xyz/。Hub21. David Huber。奇异吸引子的技术细节。 *Strange Attractors*，2021。网址：https://strangeattractors.art/details。Koo87. P Koopman。Bresenham直线绘制算法。 *Forth Dimensions*，8 6 12( )：–16，1987。Kug21. Logan Kugler。非同质化代币与艺术的未来。*tions of the ACM* 64 9 19 20, 2021 *Communica* ，（）：–。LC79. Jeff Lane 和 Loren Carpenter。用于在计算机显示器上显示参数化定义曲面的一般化扫描线算法。*and Image Processing* 11 3 290 297, 1979 *Computer Graphics* ，（）：–。Lim21. Offsetra Limited。Carbon fyi。2021。网址：https://carbon.fyi/。LPC+00. Marc Levoy、Kari Pulli、Brian Curless、Szymon Rusinkiewicz、David Koller、Lucas Pereira、Matt Ginzton、Sean Anderson、James Davis、Jeremy Ginsberg 等。数字米开朗基罗项目：大型雕像的三维扫描。载于 *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* 131 144, 2000 ，页码 – 。

MB08. Kenton McHenry 和 Peter Bajcsy。三维数据内容、文件格式和查看器概述。 *National Center for Supercomputing Applications*，1205 22，2008：。

MB21. Simon Mackenzie 和 Di¯ana B¯erzi¸na。《非同质化代币：数字事物及其犯罪生命》。*Crime, Media, Culture*，第 17416590211039797 页，2021 年。MJ17. Matt and John。CryptoPunks 项目。*Larva Labs*，2017 年。网址：https: //www.larvalabs.com/cryptopunks。MJ19. Matt and John。Autoglyphs 项目。*Larva Labs*，2019 年。网址：https: //www.larvalabs.com/autoglyphs。Nou21. Nouns。Nouns DAO 项目。2021 年。网址：https://nouns.wtf/。Pho75. Bui Tuong Phong。计算机生成图像的照明。*Communications of the ACM* 18 6 311 317, 1975 ，（）：–。

SA99. Mark Segal 和 Kurt Akeley。OpenGL 图形系统：规范（1.1 版），1999 年。say22. sayangel。使用 glTF 构建首个链上三维非同质化代币。*Blitblox*，2022 年。网址：https://mirror.xyz/angelsay.eth/Fpqj6Hawn-IWGgXm9oEYXyscIgolotYscShuNaVTmI4。Sta14. 斯坦福大学。斯坦福三维扫描资料库。2014 年。网址：http: //graphics.stanford.edu/data/3Dscanrep/。TBDI22. Jon Truby、Rafael Dean Brown、Andrew Dahdal 和 Imad Ibrahim。区块链、气候损害与死亡：减少非同质化代币和比特币碳排放、死亡率及净零排放影响的政策干预。*Energy Research & Social Science*，88:102499，2022 年。TM20. David Twomey 和 Andrew Mann。加密货币市场中的欺诈与操纵。*Corruption and fraud in financial markets: malpractice misconduct and manipulation* 624 2020 ，，，。Tok17. Render Token。Render Token（RNDR）白皮书。2017 年。访问日期：2022 年 8 月 9 日。WB82. Hans Georg Wiedemann 和 Gerhard Bayer。娜芙蒂蒂半身像。*Analytical chemistry* 54 4 619 628, 1982 ，（）：–。

WJL+20. Isaac Ronald Ward、Jack Joyner、Casey Lickfold、Stash Rowe、Yulan Guo 和 Mohammed Bennamoun。图神经网络实用指南。*arXiv preprint arXiv:2010.05234*，2020 年。WLWC21. Qin Wang、Rujia Li、Qi Wang 和 Shiping Chen。非同质化代币（NFT）：概述、评估、机遇与挑战。*arXiv:2105.07447* 2021 *arXiv preprint* ，。Wu21. Wattsy 和 untokyo。我们在网上永久保存数字艺术。*Art* 2021 *Kohi* ，。网址：https://kohi.art/onchain。Yod22. Madison Yoder。“OpenSea”式侵权：非同质化代币的知识产权影响。*and Computer Law Journal* 6 2 4, 2022 *The University of Cincinnati Intellectual Property* ，（）：。ZXD+20. Zibin Zheng、Shaoan Xie、Hong-Ning Dai、Weili Chen、Xiangping Chen、Jian Weng 和 Muhammad Imran。智能合约概述：挑战、进展与平台。*tems* 105 475 491, 2020 *Future Generation Computer Sys-* ，：–。

# 附录

## A 环境考量

Shackled 是在以太坊区块链的工作量证明时代开发的。在本文中讨论的两个主要合约库（Shackled Genesis 和 Shackled Icons）的部署及后续使用，导致能源总消耗相当于排放 54.6 公吨二氧化碳；这一数据由以太坊排放计算器 *Carbon.FYI* [Lim21]测得。

我们意识到工作量证明区块链对环境造成的影响，并通过注销由***Offsetra Limited***提供的经核证碳信用单位，抵消了相当于该数量五倍的二氧化碳排放量——273 吨。Verra 认证的已注销单位记录可通过以下链接查看：

1. https://registry.verra.org/myModule/rpt/myrpt.asp?r=206&h=162735.
2. https://registry.verra.org/myModule/rpt/myrpt.asp?r=206&h=166285.

## B 数据可用性与可复现性

数据也是 列表和代码可通过以下 链接获取：

– Shackled 网站。– Shackled 无代码用户界面网站。– 用于访问数据集的 GitHub 存储库。– 在 Etherscan 上验证的 Shackled 智能合约。

## C Shackled 计算出的首个链上渲染

由于 Shackled 是首个链上三维渲染引擎，我们认为它的第一个输出应当就是首个链上三维渲染结果。为留作纪念，我们已将该渲染结果嵌入本文（图 7）。

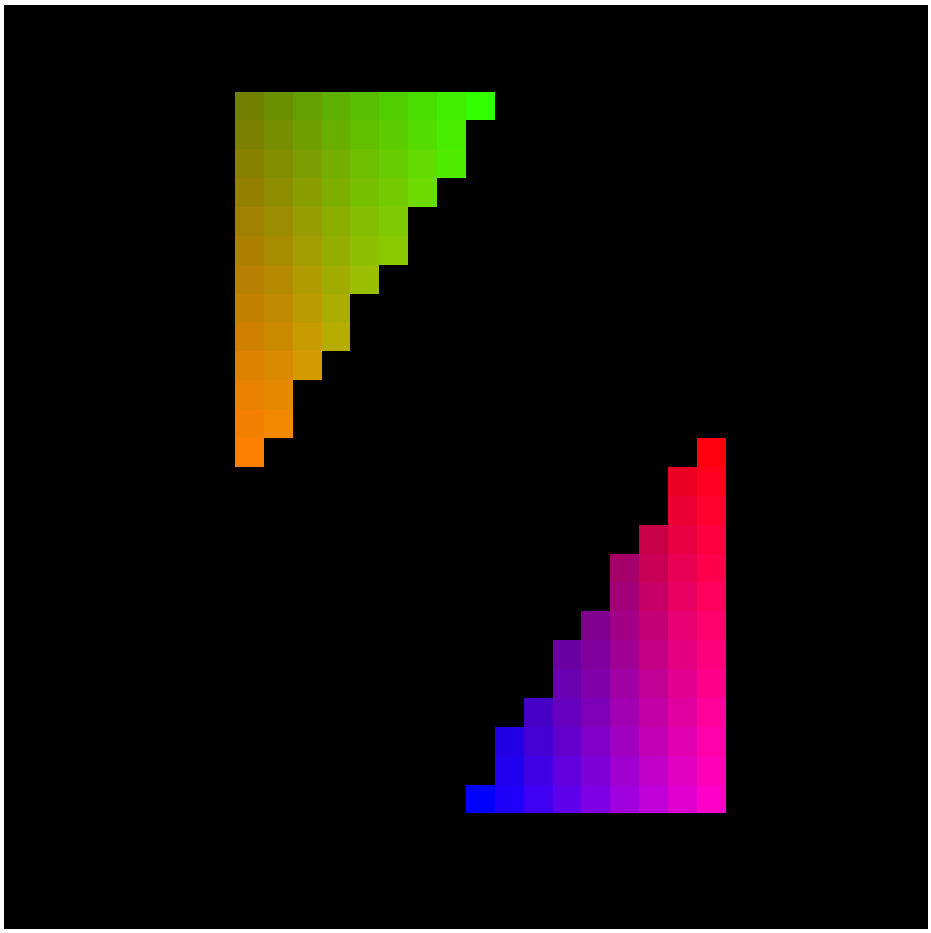


图 7：Shackled 计算出的首个链上渲染。我们认为这是**以太坊上计算出的首个链上三维渲染结果**。渲染结果中的三角形自然呈现为二维，但它们在三维空间中的定义分别由以下点构成：(-1,0,0)、(-1,2,0)、(0,2,0) 和 (1,0,5)、(1,-2,5)、(0,-2,5)。