5.1 几何建模

All Rights Reserved

大纲

1. 基础知识
2. OpenCasCade内核和STEP标准
3. 几何建模工具CadQuery
4. 上机实例

All Rights Reserved

1. 基础知识

常用二次曲面表达方式

- **隐式表达** $f(x,y,z)=0$

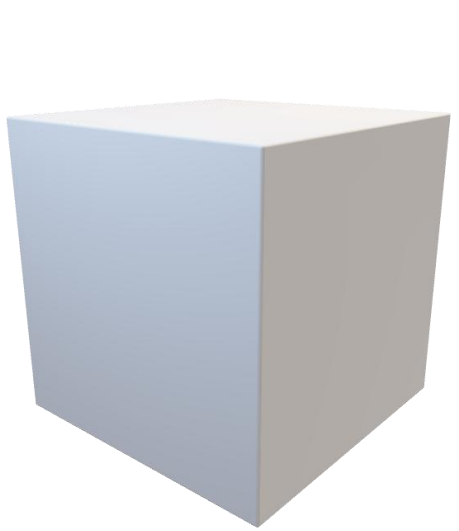
平面 $Ax+By+Cz+D=0$

球面 $x^2+y^2+z^2=R^2$

圆柱面 $x^2+y^2=R^2$

- **参数表达** $f(u,v)=(x(u,v),y(u,v),z(u,v))$

球面
$$S(\theta,\phi)=\begin{cases}x(\theta,\phi)=r\sin\theta\cos\phi\\ y(\theta,\phi)=r\sin\theta\sin\phi\\ z(\theta,\phi)=r\cos\theta\end{cases}$$



平面

圆柱面

圆锥面

球面

椭球面

常用参数自由曲线曲面 – Bezier曲线的产生

Bezier曲线可看做一种针对控制点的插值曲线，由法国雷诺汽车公司 Pierre Bezier 于 1962 年提出。

已知控制点为 $\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2$

$\mathbf{P}_{01}$ 为控制点$\mathbf{P}_0, \mathbf{P}_1$ 之间的差值点

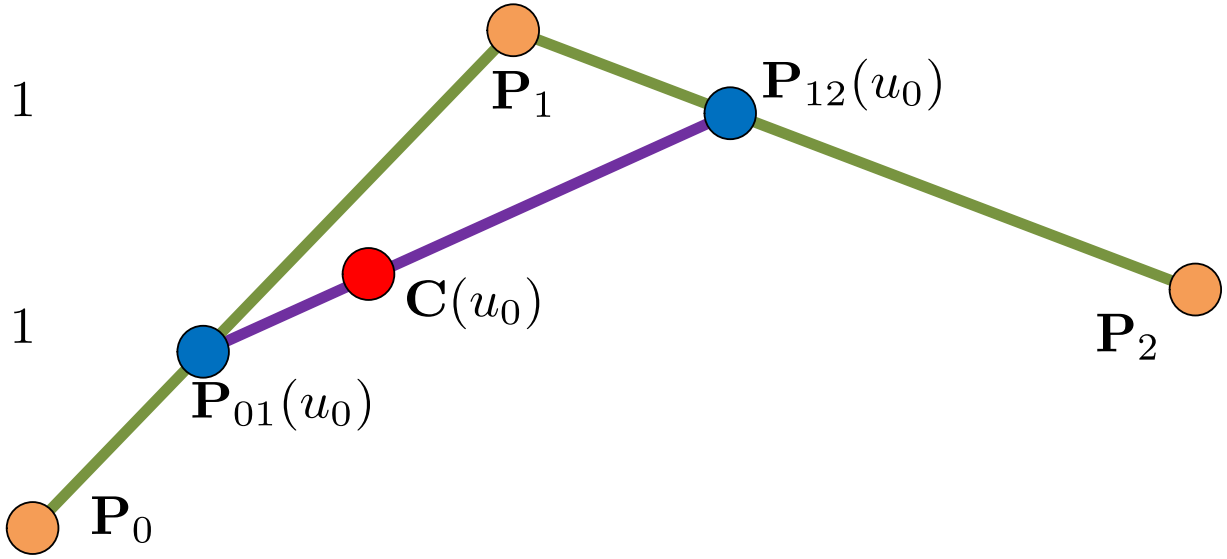
$$\mathbf{P}_{01}(u_0) = (1-u_0)\mathbf{P}_0 + u_0\mathbf{P}_1, \qquad 0 \le u_0 \le 1$$

$\mathbf{P}_{12}$ 为控制点$\mathbf{P}_1, \mathbf{P}_2$ 之间的差值点

$$\mathbf{P}_{12}(u_0) = (1-u_0)\mathbf{P}_1 + u_0\mathbf{P}_2, \qquad 0 \le u_0 \le 1$$

Bezier曲线$\mathbf{C}(u)$ 中对应的点为

$$\mathbf{C}(u_0) = (1-u_0)\mathbf{P}_{01}(u_0) + u_0\mathbf{P}_{12}(u_0)$$



3次 Bezier 曲线示意图（Degree=3）

常用参数自由曲线曲面 – Bezier曲线的参数定义

一条 n 次 Bezier 曲线 $\mathbf{C}(u)$ 可表达为

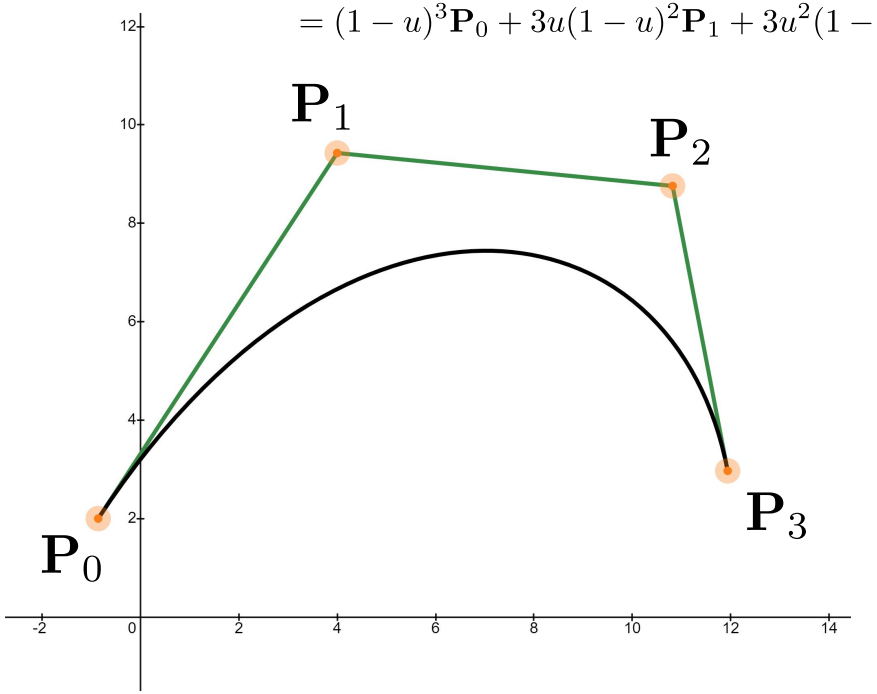
$$\begin{aligned}\mathbf{C}(u) &= (x(u), y(u), z(u)) \\ &= \sum_{i=0}^{n} B_{i,n}(u)\mathbf{P}_i \qquad 0 \le u \le 1\end{aligned}$$

其控制点为 $\mathbf{P}_i$ ，$0 \le i \le n$

其基函数是伯恩斯坦多项式 $B_{i,n}(u)$ ，定义为

$$B_{i,n}(u) = \frac{n!}{i!(n-i)!}u^i(1-u)^{n-i}$$

$$\begin{aligned}\mathbf{C}(u) &= \sum_{i=0}^{3} B_{i,3}(u)\mathbf{P}_i \\ &= (1-u)^3\mathbf{P}_0 + 3u(1-u)^2\mathbf{P}_1 + 3u^2(1-u)\mathbf{P}_2 + u^3\mathbf{P}_3\end{aligned}$$



3 次 Bezier 曲线示意图

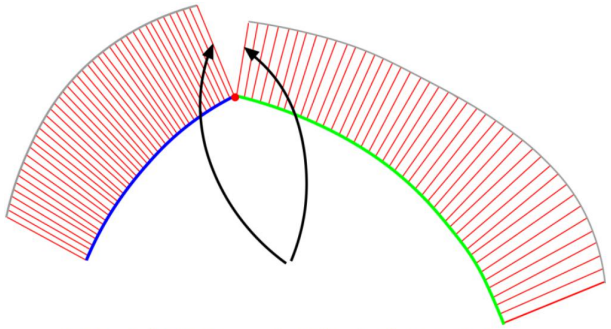
常用参数自由曲线曲面 – Bezier曲线的特点及导数

- **阶数 = 次数 + 1**，即：Order = Degree + 1
- **控制点数 = 次数 + 1**，即：Number of Control Points = Degree + 1
- 当给定的控制点数较大时，曲线的次数也对应的比较高，因此曲线会比较复杂
- 当修改一个控制点整条曲线都会改变，**无法局部修改**
- 曲线的**一阶导数** $\mathbf{C}'(u)$ 可表达为

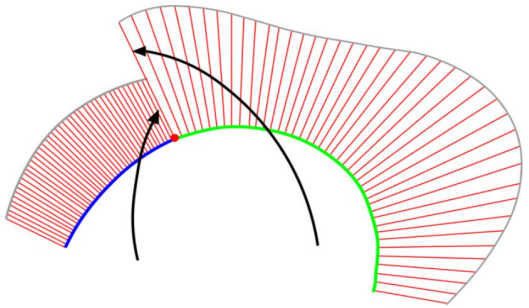
$$\begin{aligned}\mathbf{C}'(u) &= \sum_{i=0}^{n} B'_{i,n}(u)\mathbf{P}_i \\ &= n\sum_{i=0}^{n-1} B_{i,n-1}(u)(\mathbf{P}_{i+1}-\mathbf{P}_i)\end{aligned}$$

常用参数自由曲线曲面 – Bezier曲线拼接及拼接光滑性

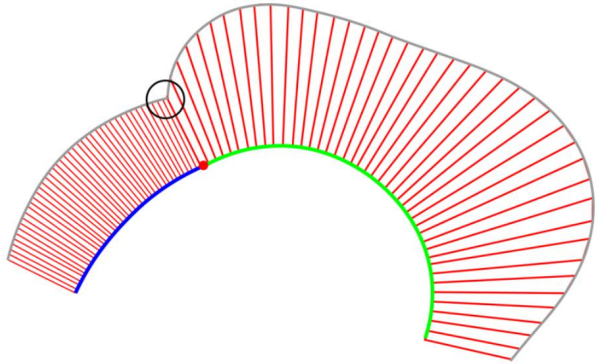
Bezier 曲线之间可以**光滑**拼接，根据光滑性要求不同，分为 **G0**、**G1**、**G2**、**G3** 连续等不同情况



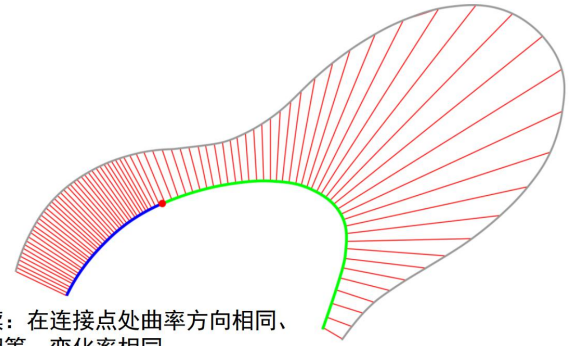
G0连续（点连续）：在连接点处曲率方向不同



G1连续（相切连续）：在连接点处曲率方向相同，大小不同



G2连续（曲率连续）：在连接点处曲率方向相同，大小相等

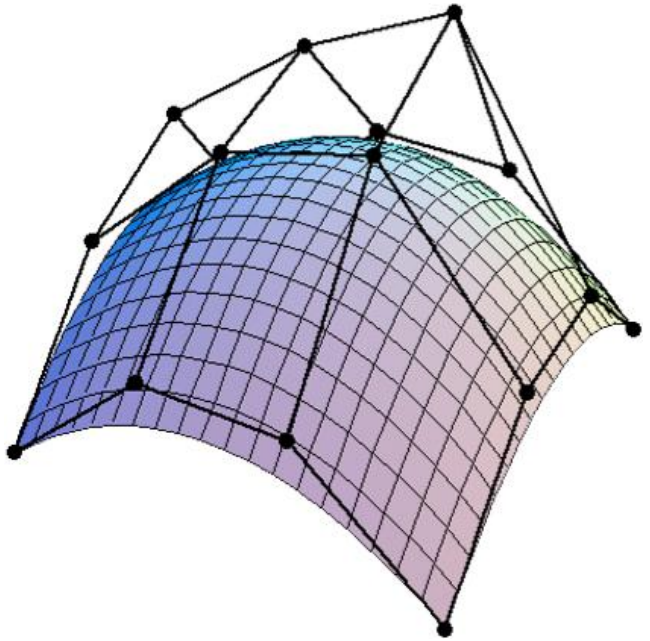


G3连续：在连接点处曲率方向相同、大小相等、变化率相同

常用参数自由曲线曲面 – Bezier曲面的参数定义

一条 $n \times m$ 次 Bezier 曲面 $\mathbf{S}(u,v)$ 可表达为

$$\mathbf{S}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_{i,n}(u)B_{j,m}(v)\mathbf{P}_{i,j}, \qquad (u,v) \in [0,1] \times [0,1]$$

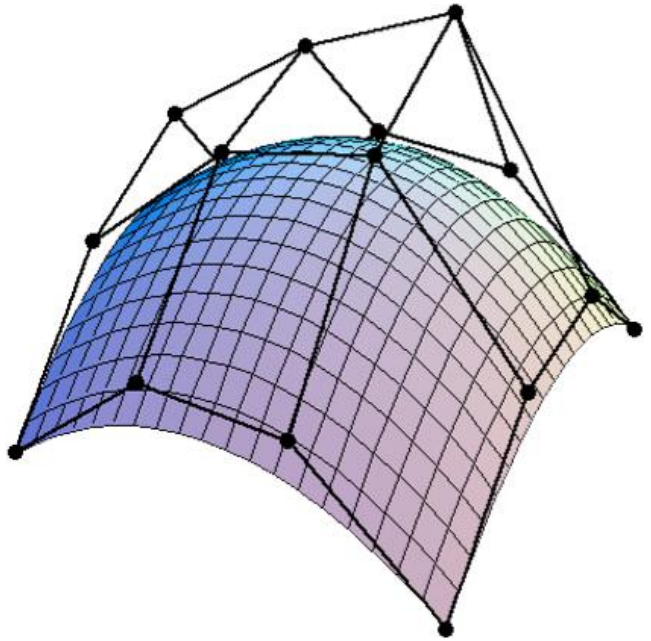


其中控制点 $\mathbf{P}_{i,j}$ $0 \le i \le n, 0 \le j \le m$ 构成了曲面的控制网格

常用参数自由曲线曲面 – Bezier曲面上的等参线

对于给定的参数u_0 ， Bezier 曲面的**等参线**是一条Bezier曲线

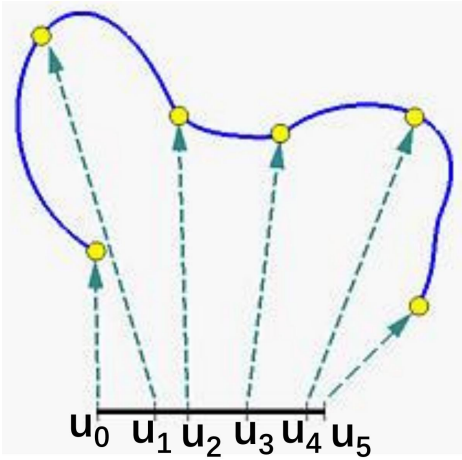
$$\begin{aligned}\mathbf{C}_{u_0}(v) &= \mathbf{S}(u_0, v) \\ &= \sum_{i=0}^{n}\sum_{j=0}^{m} B_{i,n}(u_0)B_{j,m}(v)\mathbf{P}_{i,j} \\ &= \sum_{j=0}^{m} B_{j,m}(v)\mathbf{Q}_j(u_0)\end{aligned}$$



其中

$$\mathbf{Q}_j(u_0) = \sum_{i=0}^{n} B_{i,n}(u_0)\mathbf{P}_{i,j}, \quad j = 0, 1, 2, \ldots, m$$

常用参数自由曲线曲面 – B-spline曲线的参数定义



一条 p 次 B 样条曲线 $\mathbf{C}(u)$ 可表达为

$$\mathbf{C}(u) = (x(u), y(u), z(u))$$

$$= \sum_{i=0}^{n} N_{i,p}(u)\mathbf{P}_i \qquad u_0 \le u \le u_m$$

控制点为 $\mathbf{P}_i$ ，$0 \le i \le n$

非周期单调非递减的**节点向量** $\mathbf{U}$ ， 其中包括 $m+1$ 个**节点**

$$\mathbf{U} = (u_0, u_1, u_2, \ldots, u_{m-1}, u_m)$$

$N_{i,p}(u)$ 是定义节点向量上的 **B 样条基函数**

$$N_{i,0}(u) = \begin{cases} 1, & u_i \le u < u_{i+1} \\ 0, & otherwise \end{cases}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1} + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}, \quad p \ge 1$$

常用参数自由曲线曲面 – B-spline曲线的参数特点

- **阶数 = 次数 + 1**，即：Order = Degree + 1
- **节点数 = 控制点数 + 次数 + 1**，即：Number of Knots = Number of Control Points + Degree + 1
- 改变一个控制点，只能**影响有限的区间**，区间的大小只与定义的曲线阶数相关
- 当节点向量中的节点为均匀分布时，曲线为**均匀 B 样条**（Uniform）；

  否则，曲线为**非均匀 B 样条**（Non-Uniform）

常用参数自由曲线曲面 – B-spline曲线的参数特点及其与Bezier曲线的关系

- 一般 B 样条的节点向量的起止节点为**重节点**，节点重数 = 次数 + 1，即

$$\begin{aligned}\mathbf{U} &= (u_0, u_1, u_2, \ldots, u_m, u_{m+1}) \\ &= (\underbrace{a, \ldots, a}_{p+1}, u_{p+1}, u_{p+2}, \ldots, u_{m-p-1}, \underbrace{b, \ldots, b}_{p+1})\end{aligned}$$

一般取$a = 0, \quad b = 1$

- 如果 B 样条曲线的控制点数与阶数相等，节点向量端点u_0 和u_m 的节点重数与阶数相等，

  B 样条曲线退化为 Bezier 曲线

- 如果 B 样条曲线的每个内部节点$u_i, p+1 \le i \le m-p-1$ 的节点重复度与曲线的次数相同，则 **B 样条可分解为多段 Bezier 曲线**

常用参数自由曲线曲面 – B-spline曲线的导数

一条 p 次 B 样条曲线 $\mathbf{C}(u)=\sum_{i=0}^{n} N_{i,p}(u)\mathbf{P}_i$

控制点为 $\mathbf{P}_i$ ，$0 \le i \le n$

节点向量为 $\mathbf{U}=(\underbrace{a,\ldots,a}_{p+1}, u_{p+1}, u_{p+2}, \ldots, u_{m-p-1}, \underbrace{b,\ldots,b}_{p+1})$

曲线的**一阶导数**为一条 $p-1$ 次 B 样条曲线 $\mathbf{C}'(u)=\sum_{i=0}^{n-1} N_{i,p-1}(u)\mathbf{Q}_i$

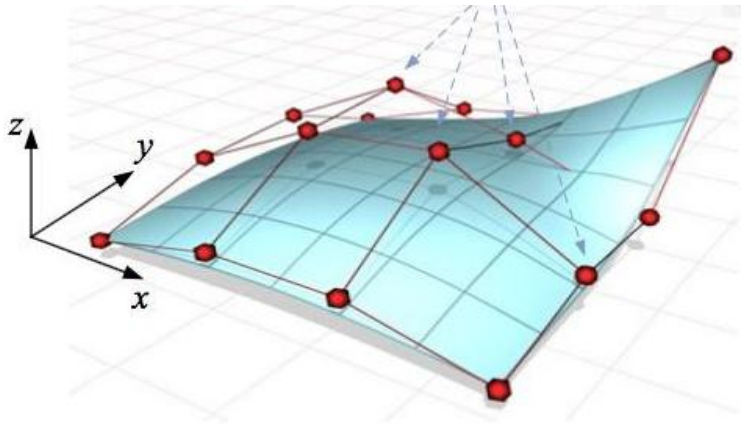
控制点为 $\mathbf{Q}_i = p\frac{\mathbf{P}_{i+1}-\mathbf{P}_i}{u_{i+p+1}-u_{i+1}}$ $0 \le i \le n-1$

节点向量为 $\mathbf{U}=(\underbrace{a,\ldots,a}_{p}, u_{p+1}, u_{p+2}, \ldots, u_{m-p-1}, \underbrace{b,\ldots,b}_{p})$

常用参数自由曲线曲面 – B-spline曲面的参数定义

一个 $p \times q$ 次具备 $(n+1) \times (m+1)$ 个控制点的 B 样条曲面 $\mathbf{S}(u,v)$ 可表达为

$$\mathbf{S}(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) \mathbf{P}_{i,j}, \qquad (u,v) \in [a,b] \times [c,d]$$



其中控制点 $\mathbf{P}_{i,j}$ $0 \le i \le n, 0 \le j \le m$ 构成了曲面的控制网格

其中节点向量 $\mathbf{U}, \mathbf{V}$ 长度分别 $r+1, s+1$ 为

$$\mathbf{U} = (\underbrace{a, \ldots, a}_{p+1}, u_{p+1}, \ldots, u_{r-p-1}, \underbrace{b, \ldots, b}_{p+1}) \qquad \text{and} \quad r = n + p + 1$$

$$\mathbf{V} = (\underbrace{c, \ldots, c}_{q+1}, v_{q+1}, \ldots, v_{s-q-1}, \underbrace{d, \ldots, d}_{q+1}) \qquad \text{and} \quad s = m + q + 1$$

常用参数自由曲线曲面 – B-spline曲面的 u 向偏导数

B 样条曲面 $\mathbf{S}(u,v)$ 对 u 的偏导数可表达为

$$\mathbf{S}_u(u,v)=\sum_{i=0}^{n-1}\sum_{j=0}^{m}N_{i,p-1}(u)N_{j,q}(v)\mathbf{P}_{i,j}^{(1,0)}$$

其中 $\mathbf{P}_{i,j}^{(1,0)}=p\frac{\mathbf{P}_{i+1,j}-\mathbf{P}_{i,j}}{u_{i+p+1}-u_{i+1}}$

其中节点向量 $\mathbf{V}$ 保持不变，节点向量$\mathbf{U}$ 为

$$\mathbf{U}=(\underbrace{a,\ldots,a}_{p},u_{p+1},\ldots,u_{r-p-1},\underbrace{b,\ldots,b}_{p})$$

常用参数自由曲线曲面 – B-spline曲面的 v 向偏导数

B 样条曲面 $\mathbf{S}(u,v)$对 v 的偏导数可表达为

$$\mathbf{S}_v(u,v)=\sum_{i=0}^{n}\sum_{j=0}^{m-1}N_{i,p}(u)N_{j,q-1}(v)\mathbf{P}_{i,j}^{(0,1)}$$

其中 $\mathbf{P}_{i,j}^{(0,1)}=q\frac{\mathbf{P}_{i,j+1}-\mathbf{P}_{i,j}}{v_{j+q+1}-v_{j+1}}$

其中节点向量 $\mathbf{U}$ 保持不变，节点向量 $\mathbf{V}$ 为

$$\mathbf{V}=(\underbrace{c,\ldots,c}_{q},v_{q+1},\ldots,v_{s-q-1},\underbrace{d,\ldots,d}_{q})$$

常用参数自由曲线曲面 – B-spline曲面的混合偏导数

B 样条曲面 $\mathbf{S}(u,v)$ 的混合偏导数可表达为

$$\mathbf{S}_{uv}(u,v)=\sum_{i=0}^{n-1}\sum_{j=0}^{m-1}N_{i,p-1}(u)N_{j,q-1}(v)\mathbf{P}_{i,j}^{(1,1)}$$

其中 $\mathbf{P}_{i,j}^{(1,1)}=q\frac{\mathbf{P}_{i,j+1}^{(1,0)}-\mathbf{P}_{i,j}^{(1,0)}}{v_{j+q+1}-v_{j+1}}$

其中节点向量 $\mathbf{U},\mathbf{V}$ 分别为

$$\mathbf{U}=(\underbrace{a,\ldots,a}_{p},u_{p+1},\ldots,u_{r-p-1},\underbrace{b,\ldots,b}_{p})$$

$$\mathbf{V}=(\underbrace{c,\ldots,c}_{q},v_{q+1},\ldots,v_{s-q-1},\underbrace{d,\ldots,d}_{q})$$

常用参数自由曲线曲面 – Bezier曲线的有理形式

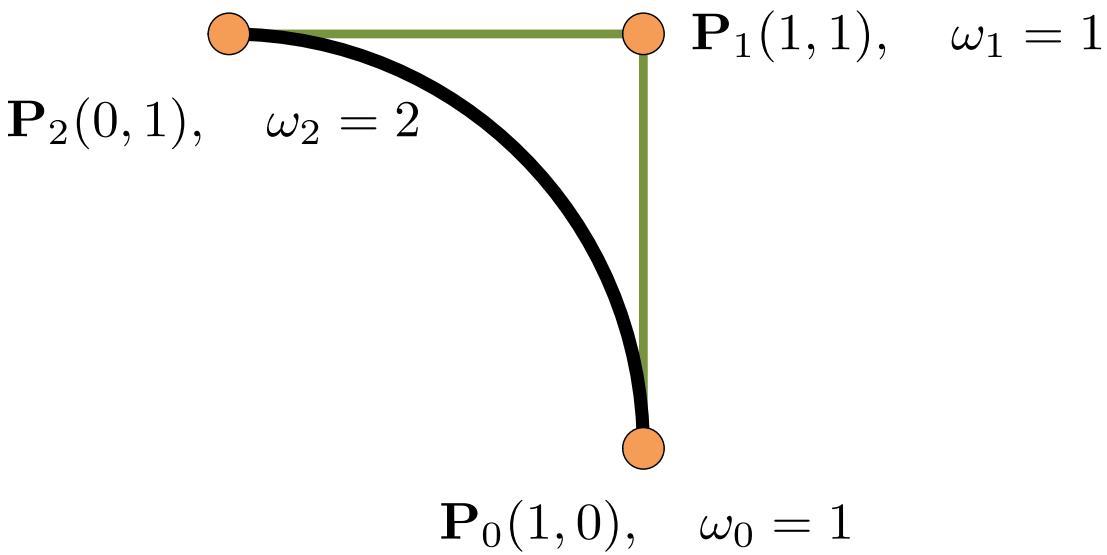
一条 n 次**有理** Bezier 曲线 $\mathbf{C}(u)$ 可表达为

$$\mathbf{C}(u)=\frac{\sum_{i=0}^{n} B_{i,n}(u)\omega_i\mathbf{P}_i}{\sum_{i=0}^{n} B_{i,n}(u)\omega_i}, \qquad 0\le u\le 1$$

其中 ω_i 为权系数

有理 Bezier 曲线可以精确表示圆弧等二次曲线

$$\mathbf{C}(u)=\frac{(1-u)^2\mathbf{P}_0+2u(1-u)\mathbf{P}_1+2u^2\mathbf{P}_2}{(1-u)^2+2u(1-u)+2u^2}, \qquad 0\le u\le 1$$



用一条2次有理Bezier曲线精确表示二次圆弧曲线

常用参数自由曲线曲面 – Bezier曲面的有理形式

一条 $n \times m$ 次**有理** Bezier 曲面 $\mathbf{S}(u,v)$ 可表达为

$$\mathbf{S}(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} B_{i,n}(u)B_{j,m}(v)\omega_{i,j}\mathbf{P}_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} B_{i,n}(u)B_{j,m}(v)\omega_{i,j}}, \qquad (u,v) \in [0,1] \times [0,1]$$

有理 Bezier 曲面可以精确表示球面等二次曲面

常用参数自由曲线曲面 –B-spline曲线的有理形式

一条 p 次**有理** B 样条曲线 $\mathbf{C}(u)$ 可表达为

$$\mathbf{C}(u)=\frac{\sum_{i=0}^{n}N_{i,p}(u)\omega_i\mathbf{P}_i}{\sum_{i=0}^{n}N_{i,p}(u)\omega_i},\qquad a\le u\le b$$

其中 ω_i 为权系数

控制点为 $\mathbf{P}_i$ ，$0\le i\le n$

节点向量 $\mathbf{U}$ ， 其中包括$m+1$ 个**节点**

$$\mathbf{U}=(\underbrace{a,\ldots,a}_{p+1},u_{p+1},u_{p+2},\ldots,u_{m-p-1},\underbrace{b,\ldots,b}_{p+1})$$

令

$$R_{i,p}(u)=\frac{N_{i,p}(u)\omega_i}{\sum_{i=0}^{n}N_{i,p}(u)\omega_i}$$

为有理基函数，则

$$\mathbf{C}(u)=\sum_{i=0}^{n}R_{i,p}(u)\mathbf{P}_i$$

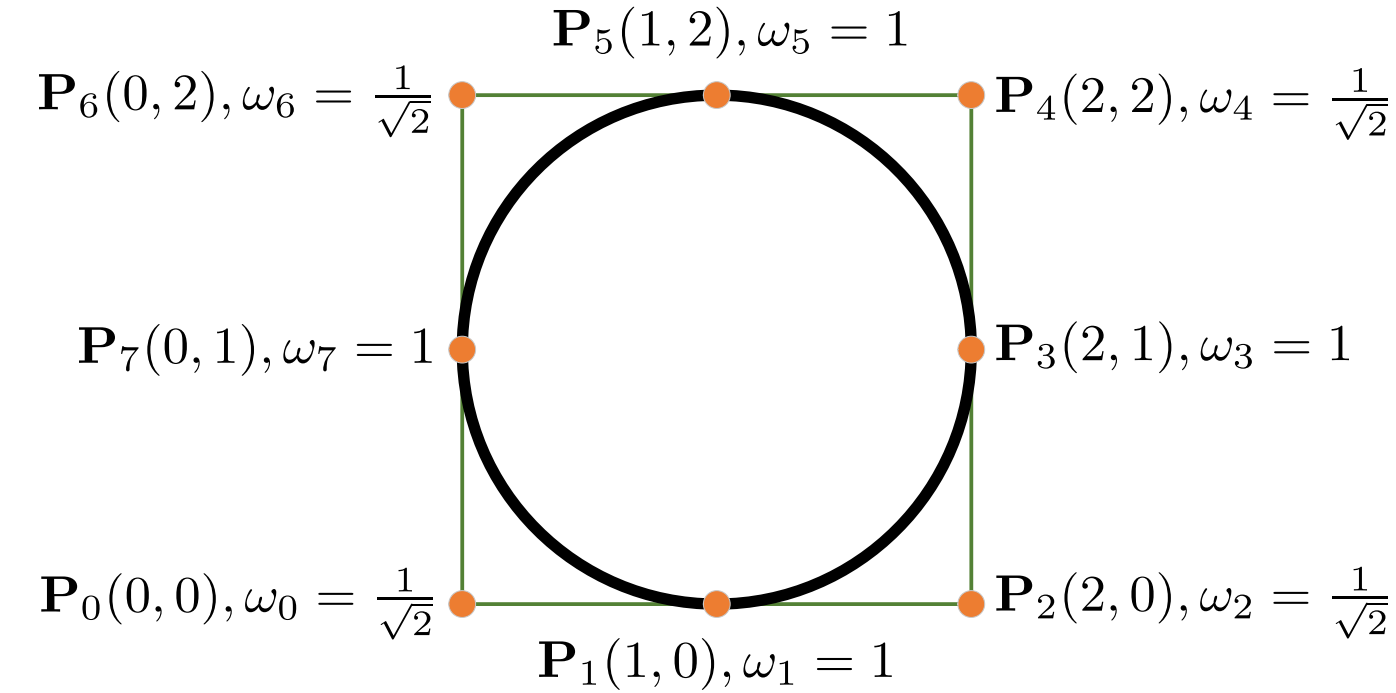
常用参数自由曲线曲面 – NURBS 曲线

- 当节点向量中的节点非均匀分布时，B 样条曲线称为 **NURBS 曲线**

  (Non Uniform Rational B-spline Curve)

- **NURBS 曲线可以精确表示圆等二次曲线**

$$\mathbf{U} = (0, 0, 0, 0.25, 0.25, 0.5, 0.5, 0.75, 0.75, 1, 1, 1)$$



常用参数自由曲线曲面 – B-spline曲面的有理形式和 NURBS 曲面

一个 $p \times q$ 次具备 $(n+1) \times (m+1)$ 个控制点的**有理** B 样条曲面 $\mathbf{S}(u,v)$ 可表达为

$$\mathbf{S}(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)\omega_{i,j}\mathbf{P}_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)\omega_{i,j}}, \qquad (u,v) \in [a,b] \times [c,d]$$

其中节点向量 $\mathbf{U}, \mathbf{V}$ 长度分别 $r+1, s+1$ 为

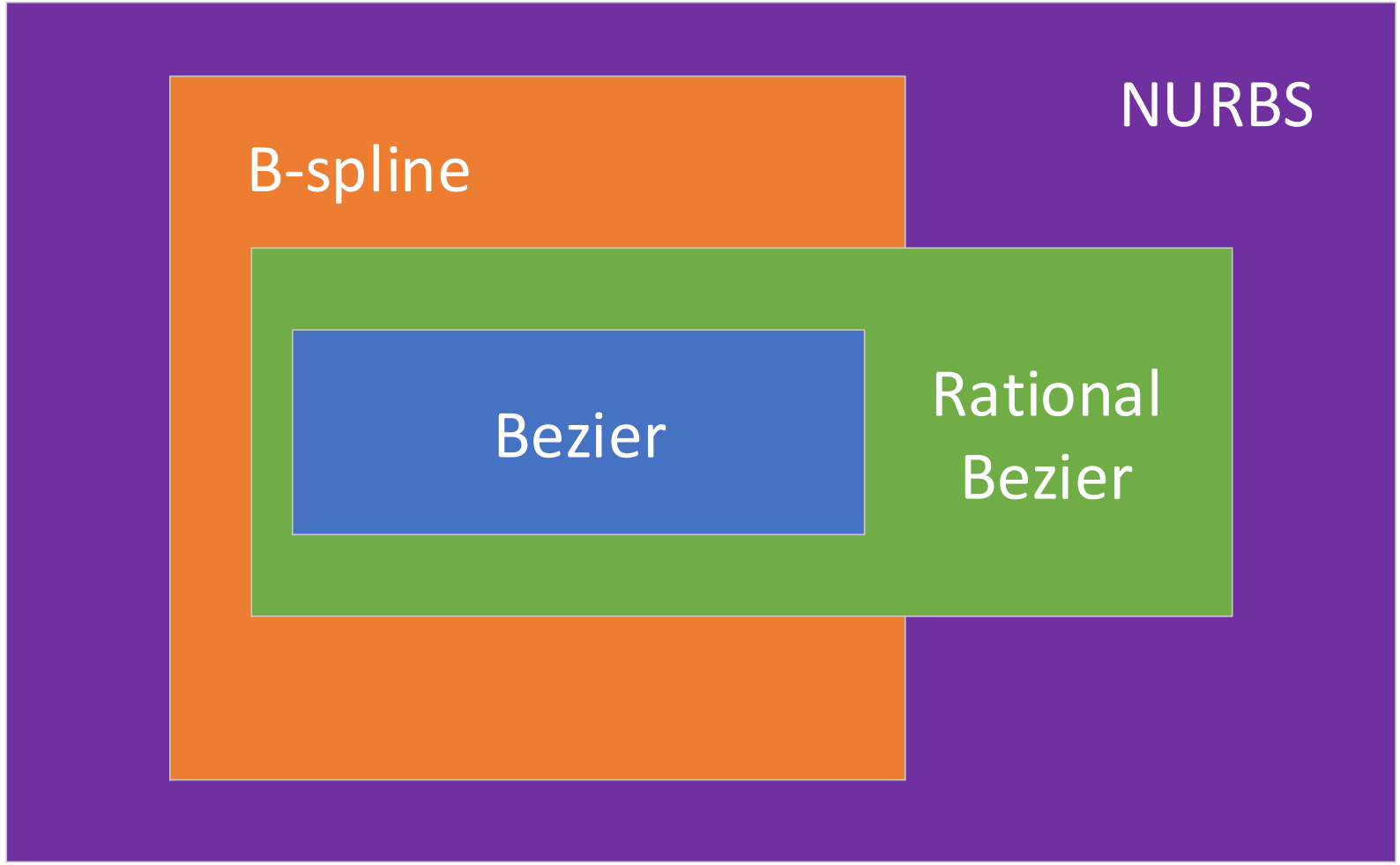
$$\mathbf{U} = (\underbrace{a,\ldots,a}_{p+1}, u_{p+1},\ldots,u_{r-p-1}, \underbrace{b,\ldots,b}_{p+1}) \quad \text{and} \quad r = n+p+1$$

$$\mathbf{V} = (\underbrace{c,\ldots,c}_{q+1}, v_{q+1},\ldots,v_{s-q-1}, \underbrace{d,\ldots,d}_{q+1}) \quad \text{and} \quad s = m+q+1$$

- 当节点向量中的节点非均匀分布时，B 样条曲面称为 **NURBS 曲面**

(Non Uniform Rational B-spline Surface)

常用参数自由曲线曲面 – Bezier、B-spline与NURBS之间的关系



样条曲线的节点插入（Knot Insertion）

设一条曲线 $\mathbf{C}(u)=\sum_{i=0}^{n}N_{i,p}(u)\mathbf{P}_i$ 是定义在节点向量 $\mathbf{U}=(u_0,u_1,u_2,\ldots,u_{m-1},u_m)$ 上的样条曲线

设 $\overline{u}\in[u_k,u_{k+1})$ 在 $\mathbf{U}$ 中的节点重复度为 s，要将 $\overline{u}$ 插入 $\mathbf{U}$ 中 r 次，使其节点重复度成为 $\mathrm{r+s}$ ·

要求**节点插入后只允许改变空间基底，不允许改变曲线在几何形状和参数化方式**，则共需要插入 r 个控制点

记第 i 个新控制点为 $\mathbf{Q}_{i,r}$，且 $\mathbf{Q}_{i,0}=\mathbf{P}_i$

$$\mathbf{Q}_{i,r}=\alpha_{i,r}\mathbf{Q}_{i,r-1}+(1-\alpha_{i,r})\mathbf{Q}_{i-1,r-1}$$

$$\alpha_{i,r}=\begin{cases}1, & i\le k-p+r-1\\ \dfrac{\overline{u}-u_i}{u_{i+p-r+1}-u_i}, & k-p+r\le i\le k-s\\ 0, & i\ge k-s+1\end{cases}$$

样条曲线的节点细化（Knot Refinement)

设一条曲线 $\mathbf{C}(u)=\sum_{i=0}^{n}N_{i,p}(u)\mathbf{P}_i$ 是定义在节点向量 $\mathbf{U}=(u_0,u_1,u_2,\ldots,u_{m-1},u_m)$ 上的样条曲线

设 $\mathbf{X}=\{x_0,x_1,\ldots,x_r\}$ ，对所有的 i 满足 $x_i\le x_{i+1},u_0\le x_i\le u_m$ 。把 $\mathbf{X}$ 中的元素插入 $\mathbf{U}$ 中，

记第 i 个新控制点为 $\mathbf{Q}_i,i=0,1,\ldots,n+r+1$ r

要求**节点细化后只允许改变空间基底，不允许改变曲线在几何形状和参数化方式**

则无需额外的数学推导既可实现

方法一：通过多次应用节点插入算法实现。

方法二：可参考 The NURBS Book 查阅更高效的工程实现方法。

将 NURBS 样条曲线分解为 Bezier 形式

由于样条曲线的每个内部节点 $u_i, p+1 \le i \le m-p-1$ 的节点重复度与曲线的次数相同时，**样条曲线可分解为多段 Bezier 曲线**，

因此可利用**节点插入**或**节点细化**的方法，将一条 B-spline 曲线分解为多段 Bezier 曲线，或将一条 NURBS 曲线分解为多段有理 Bezier 曲线。

样条曲面的节点插入和节点细化

设一条曲面 $\mathbf{S}(u,v)=\sum_{i=0}^{n}\sum_{j=0}^{m}N_{i,p}(u)N_{j,q}(v)\mathbf{P}_{i,j}$ 是定义在节点向量 $\mathbf{U}\times\mathbf{V}$ 上的样条曲面

对节点向量$\mathbf{U}$ 的节点插入（节点细化）可通过对$m+1$ 列控制点应用曲线的节点插入（节点细化）算法实现

对节点向量$\mathbf{V}$ 的节点插入（节点细化）可通过对$n+1$ 行控制点应用曲线的节点插入（节点细化）算法实现

插值问题与拟合问题

给定n个离散点$\mathbf{P}=\{\mathbf{P}_0,\mathbf{P}_1,\ldots,\mathbf{P}_{n-2},\mathbf{P}_{n-1}\}$，插值与拟合就是要通过这些离散点，确定某一类已知曲线（或曲面）的参数或寻求某个近似曲线（或曲面），使所得到的近似曲线（或曲面）与离散点之间具备较高的拟合精度。

- 如果要求这个近似曲线（或曲面）经过所已知的所有离散点，则称此类问题为**插值问题**。
- 如果不要求近似曲线（或曲面）通过所有离散点，而是要求它能较好地反映数据变化规律的近似曲线（或曲面）的方法称为**拟合问题**。

求交问题的分类

- 求交问题是计算几何学、几何造型与设计、工业分析与制造应用的基本问题。
- 求交问题最关注的是**高效率**求解和描述几何建模所需的**高精度交点（交线）**的所有信息。
- 根据求交元素的类型不同，可分为三个类别的求交问题：
  - 点 – 点、点 – 线、点 – 面
  - 线 – 线、线 – 面
  - 面 – 面

曲面求交方法的分类

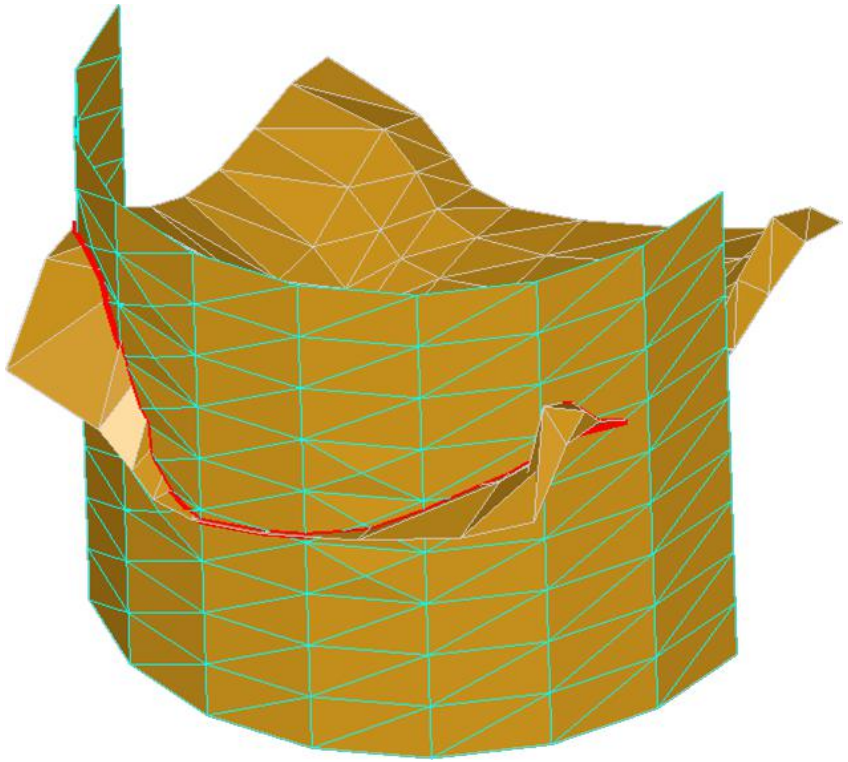
- 常用的求交思路可分为四类
  - 解析法
  - 离散法
  - 分割法（细分法）
  - 追踪法
- 单一的求交算法都有其各自的优缺点和适应范围，没有一个通用的健壮性算法。
- 普遍的做法是结合多种算法，综合权衡算法的精度、效率和健壮性，利用混合方法解决问题。

曲面求交 – 解析法

- 将曲面转换为代数方程形式，然后求解高次非线性方程组获得交线。
- 只适用于低次(二次以下)曲面或多项式曲面的求交问题。
- 三次以上的自由曲面，特别是有理曲面的求交，交线代数方程的阶数将非常高。
  - 例如，一张双三次曲面片转化的代数方程为一含有324项的18次方程。

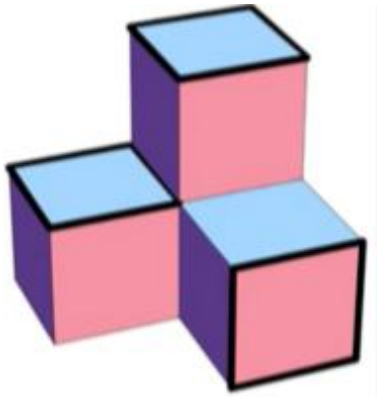
曲面求交 – 离散法

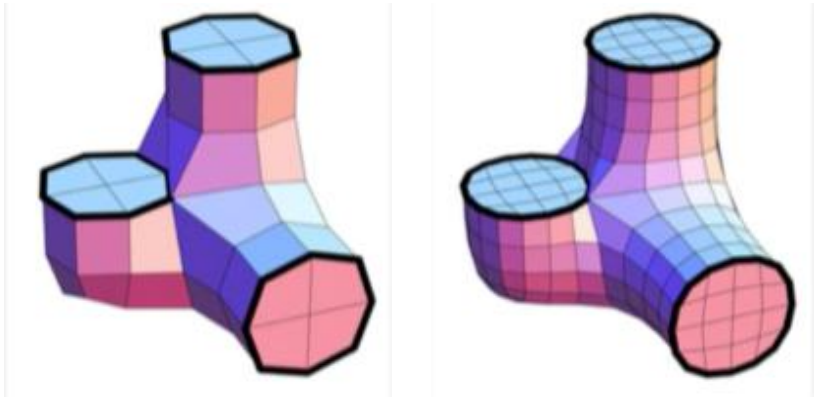
- 将两个原始光滑曲面离散成一系列的三角面片，将原本的参数曲面求交问题转化为平面三角形面片之间的相交运算，进一步的，将问题转化为直线段（第一个曲面三角形的边）和平面（第二个曲面三角形所在的面）相交的问题
- 最终将求解非线性方程组问题转化为线性问题进行求解，计算获得多个离散交点
- 将离散交点按顺序连接成的折线段，或拟合成一条连续光滑的交线即为曲面的交线

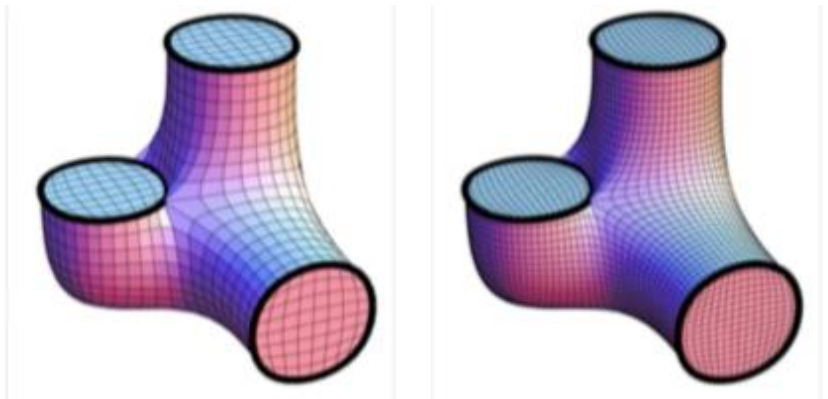


曲面求交 – 分割法（细分法）

- 利用凸包、长方体包围盒（轴对齐包围盒、方向包围盒等）表达曲面
- 如果凸包相交，则进一步分割两曲面，将父曲面四分成四个更小的子曲面，重新构造各个子曲面的凸包进行相交测试
- 重复此过程，直至曲面分割到一定程度或凸包足够小时，此时只需要对相交的凸包所包含的子曲面片进行求交既可

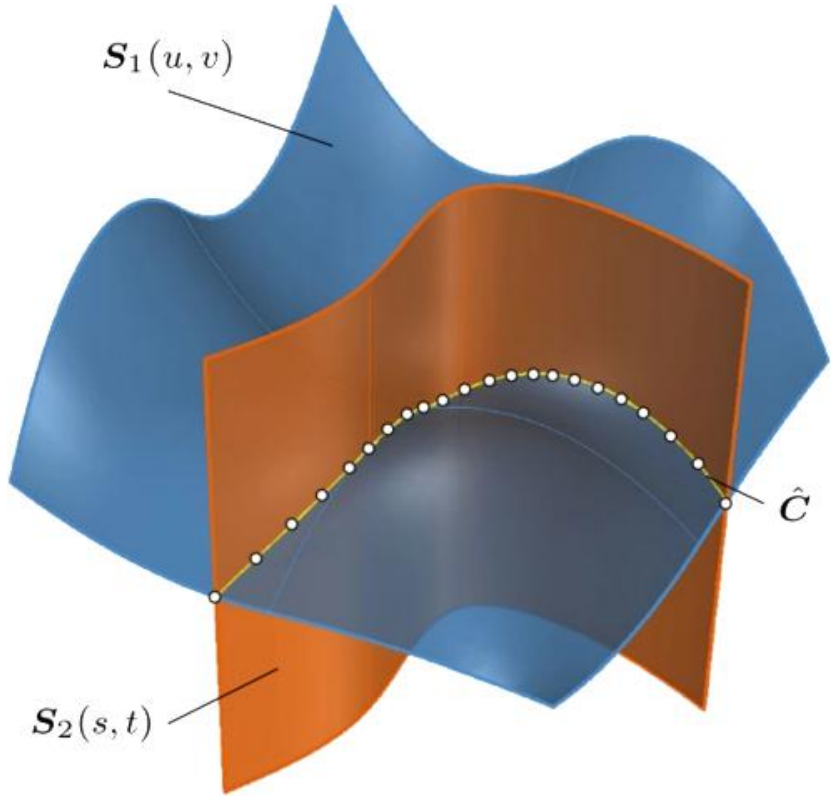






曲面求交 – 追踪法

- **搜索初始点**：两个曲面相交可能会产生多条交线，以每条交线分支上的稀疏交点作为初始点
- **追踪**：通过初始交点求后续交点
  - 计算步向量：一般用交点处两个平面的切平面法向量的叉积方向。
  - 计算步长：可采用固定步长，也可基于曲率自适应确定步长。
  - 将近似交点迭代到精确交点上：牛顿迭代法
- **排序**：将各分支上的离散交点按顺序连接，或拟合成一条连续光滑的交线

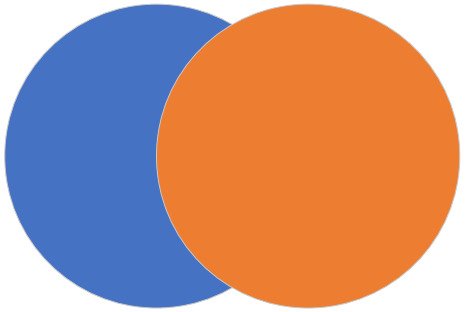


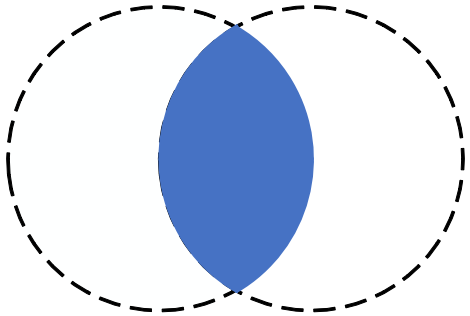
曲面求交 – 不同方法的对比优缺点

求交方法	适用范围	优点	缺点
解析法	低次曲面或多项式曲面求交	方法简单，易实现	适用范围窄
离散法	精度要求不高的求交	原理简单，易实现，适用范围广	精度不高，内存开销大
分割法	精度要求不高的求交	原理简单，易实现，适用范围广	精度低，易产生漏交、波纹现象
追踪法	任意参数曲面的求交	稳定性高，精度高，适用范围广	初始点获取、边界处理难度大，易漏交，速度慢

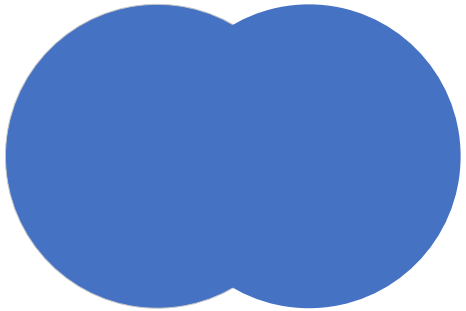
布尔运算

- 布尔运算输入两个或多个对象，输出对象之间的某种组合。
- 组合形式一般包括三种：交、并、差
- 布尔运算步骤
  - 计算输入多个对象之间的交线
  - 根据交线，将每个对象分割为多个子部分
  - 根据组合形式，确定子部分之间的组合

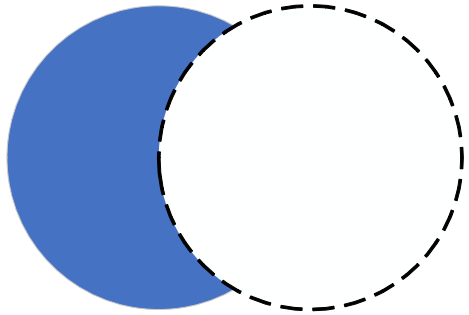




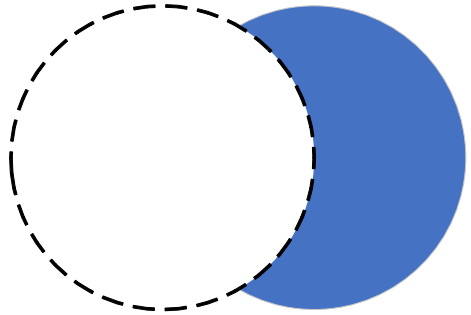
布尔交



布尔并



布尔差



布尔差

几何形体的数据表达形式 – 线框模型、表面模型、实体模型

线框模型

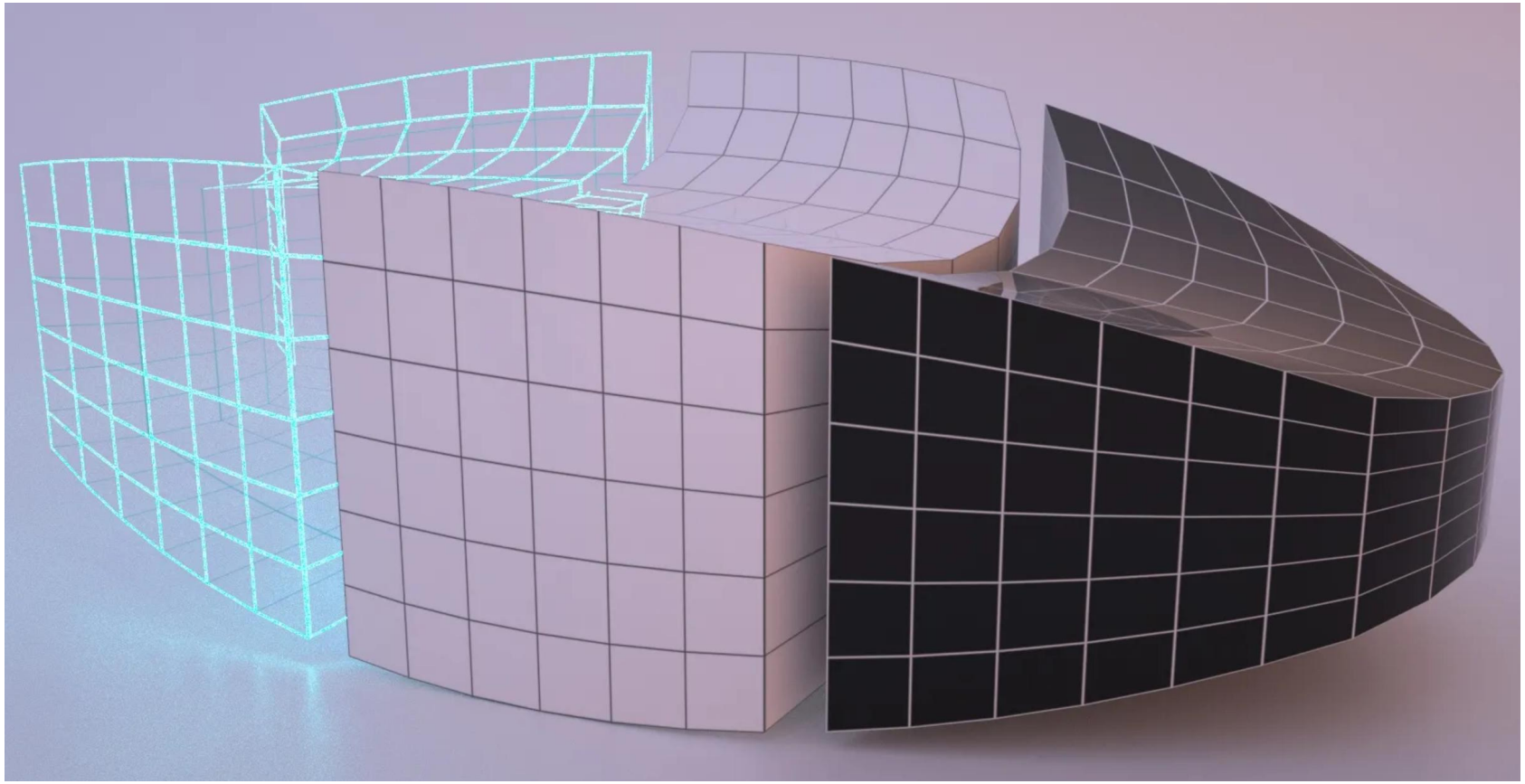
表达模型边界

表面模型

表达模型表面的曲面特征（空心）

实体模型

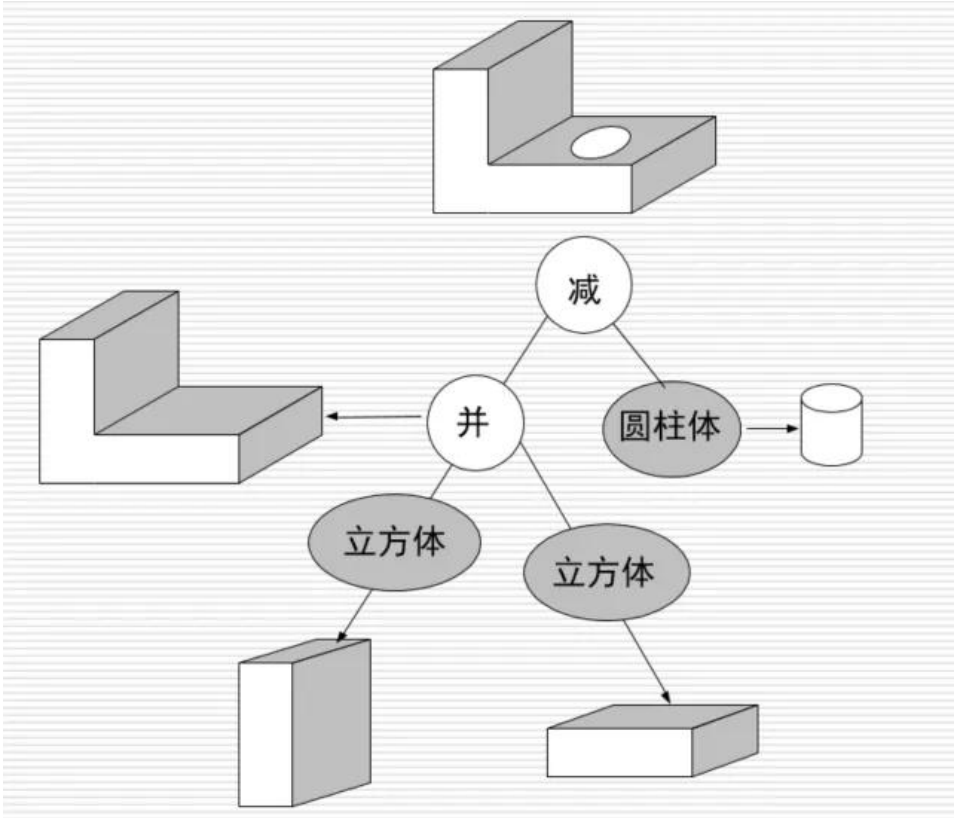
表达模型的表面和内部特征（实心）



实体模型数据表达形式 – CSG

CSG（Constructive Solid Geometry）体素构造表示法：一个物体被表示为一系列简单的基本物体（如立方体、圆柱体、圆锥体等）的布尔操作的结果，数据结构为树状结构。

- CSG树记录了实体构造过程中每步所使用的布尔运算信息
- CSG树没有详细几何信息，必须转化为其它形式才能对点、边、面等信息进行查询和编辑

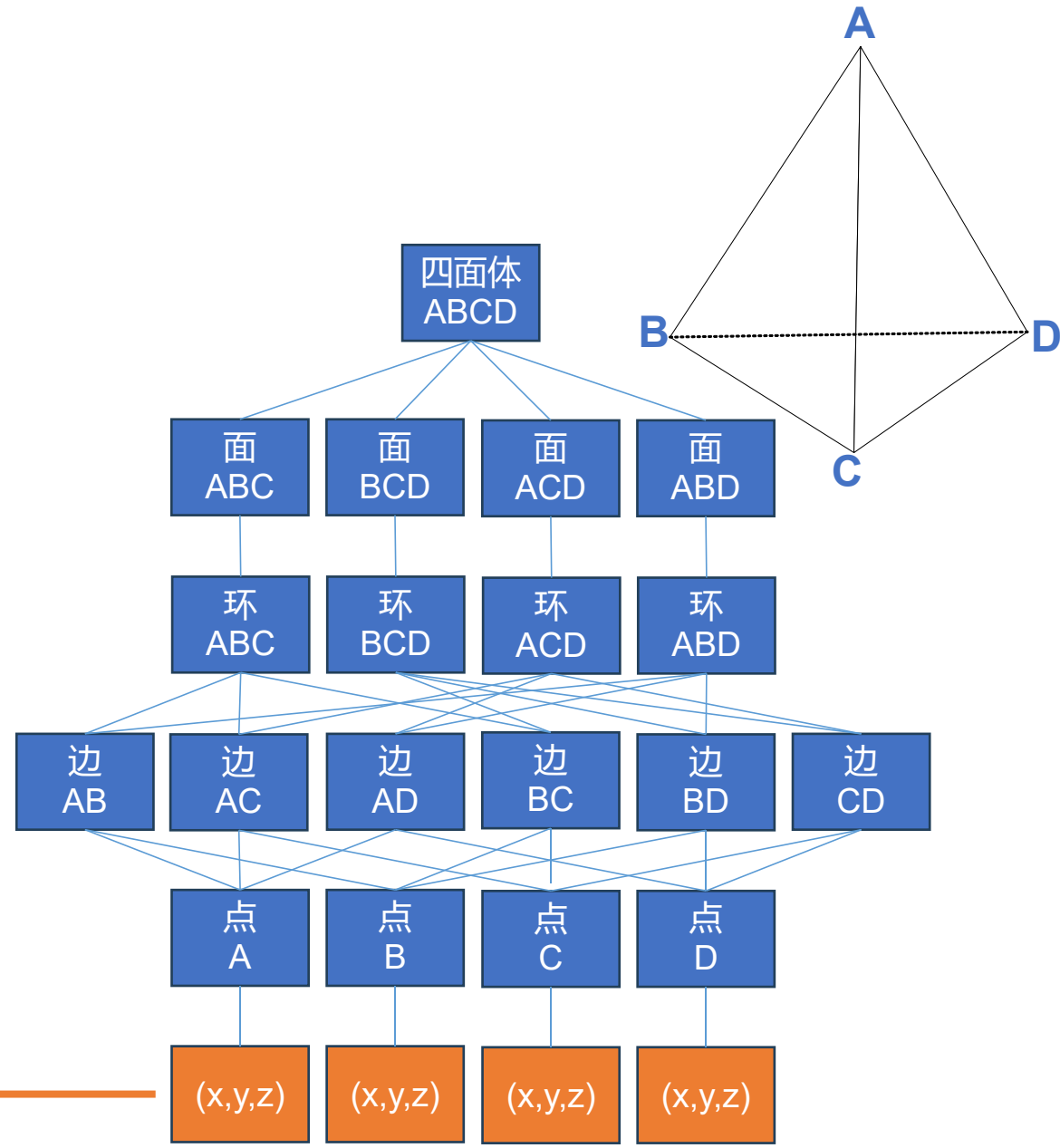


实体模型数据表达形式 – BRep

Brep（Boundary Representation）边界表示法：用一组封闭的面组成实体模型，按照**体 - 面 - 环 - 边 - 点**的层次，详细记录构成形体的所有几何元素的几何信息（Geometry）及其相互连接的拓朴关系（Topology）。

拓朴信息描述形体上的顶点、边、面的连接关系，它形成物体边界表示的“骨架”

几何信息描述形体的大小、尺寸、位置和形状等，犹如附着在“骨架”上的肌肉



2. OpenCasCade内核和STEP标准

Open CASCADE

Open CASCADE Technology SDK (OCCT)是一个采用C++开发的开源的CAD内核库，是世界上最重要的几何造型基础软件平台之一。

OCCT 使用 BRep（边界表示）方法来表示 3D 模型，并提供了 3D 曲面和实体建模、CAD 数据交换和可视化等服务，常用于 3D 建模 (CAD)、数值模拟 (CAE) 、辅助制造 (CAM)的软件开发。

OCCT是一个开放源码CAD 内核，对主流CAD 数据格式提供支持（STEP/STL/IGES 等），提供高级建模函数（拟合，有理样条曲线，拉伸、旋转、扫出、层叠拉伸、圆角、倒角、薄壳、修剪、偏移等），参数化模型，提供几何模型的特征提取，对Visual C++/MFC 有很好的支持。

https://www.opencascade.com/

All Rights Reserved

OCC历史发展

CAD技术从60年代诞生以来，经历了二维绘图、线框模型、自由曲面模型、实体造型、特征造型等重要发展阶段。随着CAD技术的发展也诞生了许多成熟和知名的CAD引擎和软件。OCCT由一家位于法国的同名公司OPEN CASCADE SAS支持和开发，

发展历史如下：

- 1980年，代初由 Matra Datavision 公司开发的 EUCLID CAD 系统的数学内核是OCCT的前身。
- 1993年，发布了一个名为 CAS.CADE（Computer Aided Software for Computer Aided Design and Engineering）的平台。
- 1996年，正式发布EUCLID QUANTUM 系统。
- 1998年，Matra Datavision 公司改变了商业模式，专注于提供软件服务。
- 1999年，Matra Datavision 的管理层做出了一个历史性的决定，即开放 CAS.CADE 库的源代码，当时称为 Open CASCADE。
- 2000年，Matra Datavision 创建了一家名为 OPEN CASCADE SAS 的附属公司，以提供基于新几何内核的支持和服务。
- 2003年，Matra Datavision 被 IBM 公司收购，而 OPEN CASCADE SAS 被另一家法国公司 Principia 收购。

All Rights Reserved

OCC许可证和LGPL

OCCT在遵循GNU宽通用公共许可证（GNU Lesser General PublicLicense，简称LGPL）。

LGPL对使用OCCT有一些强制性的要求，需要注意以下几点：

1) LGPL许可要求对用户可见，在软件使用OCCT时必须将该声明放在版权声明部分或者文档等用户可见位置。
2) 如果修改了OCCT代码，必须让应用中所用的OCCT代码版本对用户可见，必要时提供该应用编译该代码的方法，保证用户可以在LGPL许可下应用修改后的OCCT。
3) 如果不想遵循LGPL许可，那么可联系Open CASCADE公司购买商业许可。

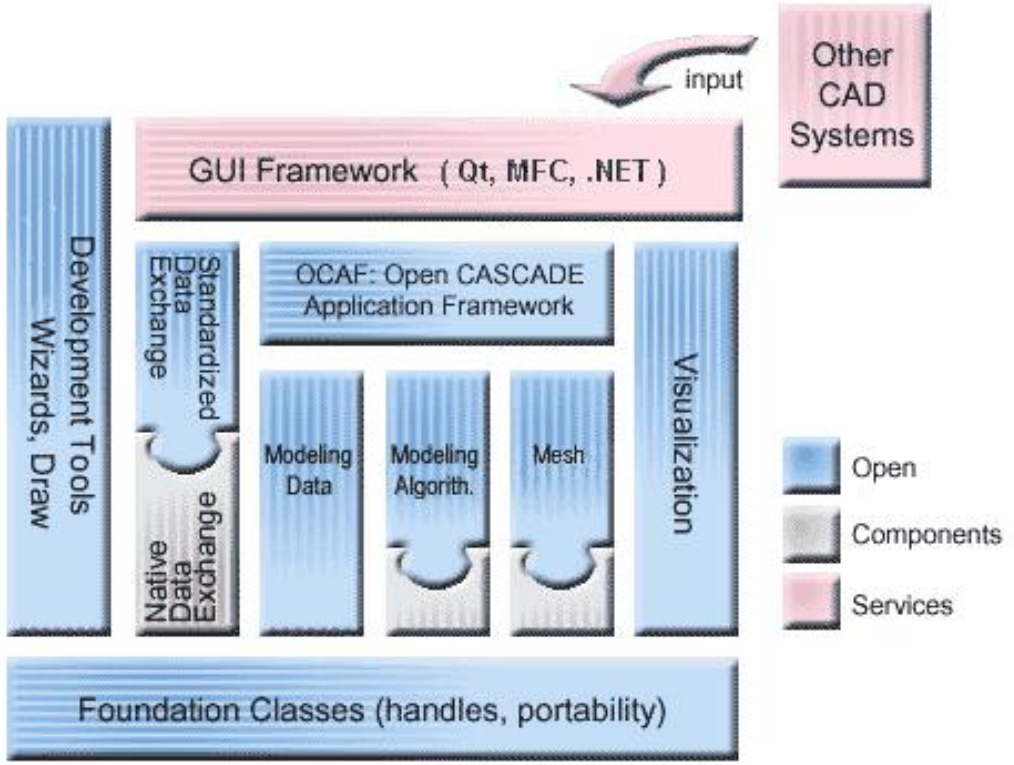
在任何许可情况下，Open CASCADE公司都不对OCCT的任何问题负责，不承担任何风险。

All Rights Reserved

OCCT的架构设计

OCCT整个架构分为7大模块

1) 基础类模块：是所有其他 OCCT 类的基础
2) 数据模块：提供数据结构以将 2D 和 3D 几何图元及其组合表示为 CAD 模型
3) 算法模块：包含大量的几何和拓扑算法；“建模算法”模块中的网格工具包实现了对象的镶嵌表示
4) 可视化模块：为图形数据表示提供了复杂的机制
5) 数据交换模块：与流行的数据格式互操作，并依靠几何体修复来提高不同供应商的 CAD 软件之间的兼容性
6) 应用程序框架模块：提供现成的解决方案，用于处理特定于应用程序的数据（用户属性）和常用功能（保存/恢复、撤消/重做、复制/粘贴、跟踪 CAD 修改等）
7) Open CASCADE Test Harness模块，也称为 Draw，提供了入口，可用作其模块的测试工具



All Rights Reserved

OCC基础模块

基础类模块包含了被高级Open CASCADE Technology类使用的数据结构和服务：

- 原始类型，如布尔型、字符型、整数型或实数型
- 处理ASCII和Unicode字符串的字符串类
- 处理静态或动态大小的数据聚合（如数组、列表、队列、集合和哈希表）的集合类
- 提供常用的数值算法和基本线性代数计算（向量和矩阵的加法、乘法、转置，解线性系统等）的类
- 表示物理量和支持日期和时间信息的基本类型
- 提供基本几何和代数实体的实现，用于定义和操作基本数据结构的原始几何类型
- 描述程序正常执行被中止情况的异常类

该模块还提供了各种通用服务，例如：

- 安全处理动态创建的对象，确保自动删除未引用的对象（智能指针）
- 可配置的优化内存管理器，提高密集使用动态创建对象的应用程序的性能
- 扩展的运行时类型信息（RTTI）机制，维护完整的类型层次结构，并提供迭代访问的手段
- 封装C++流
- 通过特定分配器自动管理堆内存
- 基本表达式解释器，便于创建定制的脚本工具、通用表达式定义等
- 处理配置资源文件和可定制消息文件的工具，便于应用程序支持多语言
- 进度指示和用户中断界面，使得低级算法能够以通用和便捷的方式与用户进行通信

All Rights Reserved

OCC数据模块

建模数据模块提供了表示2D和3D几何模型的数据结构。

建模数据模块提供了实现对象在3D中的边界表示（BRep）的数据结构。在BRep中，几何体被表示为几何体在拓扑结构中的聚合。几何体被理解为对几何体的数学描述，例如曲线和曲面（简单或规范的贝塞尔曲线、NURBS等）。拓扑是将几何对象绑定在一起的数据结构。

几何类型和工具提供几何数据结构和服务，用于：

- 描述点、向量、曲线和曲面
  - 它们在3D空间中的定位，使用轴或坐标系统
  - 它们的几何变换，通过应用平移、旋转、对称、缩放变换及其组合
- 通过插值和逼近创建参数化曲线和曲面
- 直接构造算法
- 将曲线和曲面转换为NURBS形式
- 计算2D和3D曲线上的点坐标
- 计算几何对象之间的极值

拓扑定义了简单几何实体之间的关系。一个几何体是基本的拓扑实体，它可以被分成组件（子几何体）：

- 顶点 - 对应于一个点的零维几何体
- 边 - 对应于一条曲线，由两个端点的顶点界定
- 线 - 由边按顺序相连而成
- 面 - 由线界定的平面的一部分（在2D中）或曲面（在3D中）
- 壳 - 由其线边界的边界连接的面的集合
- 实体 - 由壳界限的有限闭合3D空间的一部分
- 复合实体 - 由面的壳边界连接的实体的集合

复杂的几何体可以定义为较简单实体的组合。

All Rights Reserved

OCC建模算法模块1-底层算法

底层几何算法：

- 计算两条曲线、曲面或一条曲线与一条曲面的交点
- 将点投影到二维和三维曲线上，将点投影到曲面上，将三维曲线投影到曲面上
- 根据约束构造直线和圆
- 根据约束（插值、近似、蒙皮、间隙填充等）构建自由曲线和曲面

底层拓扑算法：

- 密铺
- 检查几何体的正确性
- 确定几何体的局部和整体属性（导数、质量惯性属性等）
- 执行仿射变换
- 查找边缘所在的平面
- 将几何体转换为 NURBS 表示
- 将单独的拓扑元素（面和边）缝合在一起形成相互连接的拓扑

All Rights Reserved

OCC建模算法模块2-高级算法

基本几何体的构造：
- 盒子
- 棱柱
- 圆柱体
- 圆锥体
- 球体
- 圆环体

基于运动的建模：
- 棱柱 - 线性扫掠
- 旋转体 - 旋转扫掠
- 管道 - 一般形式的扫掠
- 放样

布尔运算：允许从源形状的组合中创建新的形状。对于两个形状S1和S2：
- 共有（Common）包含S1和S2中都存在的所有点
- 融合（Fuse）包含S1或S2中的所有点
- 剪切（Cut）包含S1中存在但不在S2中的所有点。

局部修改的算法
- 中空（Hollowing）
- 抽壳（Shelling）
- 使用倾斜角度创建锥形形状
- 边缘的倒角和斜角

用于创建机械特征的算法
- 凹陷（Depressions）
- 凸起（Protrusions）
- 加筋（Ribs）
- 凹槽（Grooves）
- 槽（Slots）
- 平面
- 旋转面

All Rights Reserved

OCC建模算法模块3-网格相关算法

网格模块提供了如下算法功能:

网格模块提供了使用三角面片表示对象的功能，包括:

- 用于存储与形状相关的表面网格数据的数据结构和处理它们的基本算法;
- 从BRep对象（形状）构建三角面片网格的数据结构和算法;
- 用于显示具有关联的预处理和后处理数据（标量或矢量）的网格的工具。

Open CASCADE Technology包括两个网格转换器:

- VRML转换器将Open CASCADE形状转换为VRML 1.0文件（虚拟现实建模语言）。有两种表示模式可供选择：着色模式，将形状表示为由网格算法计算的三角形集合；线框模式，将形状表示为曲线集合。
- STL转换器将Open CASCADE形状转换为STL文件。STL（STL文件格式）是广泛用于快速原型制作（3D打印）的格式。

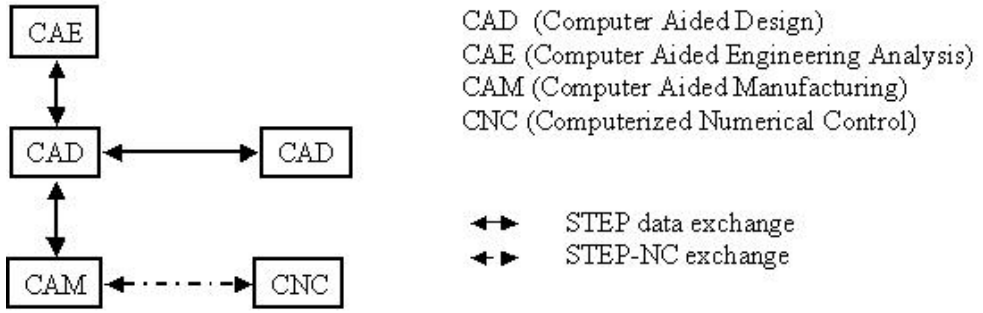
All Rights Reserved

什么是STEP标准?

在设计和制造领域，技术产品数据的管理常因系统间数据格式不同而面临重复输入和数据冗余的问题，尤其在制造业中，复杂的3D设计数据增加了操作错误和误解的风险。

据美国国家标准研究院估计，数据不兼容问题对制造业造成的损失高达900亿美元。为解决这一问题，人们提出了多种解决方案，其中最有效的是实施数据交换标准。

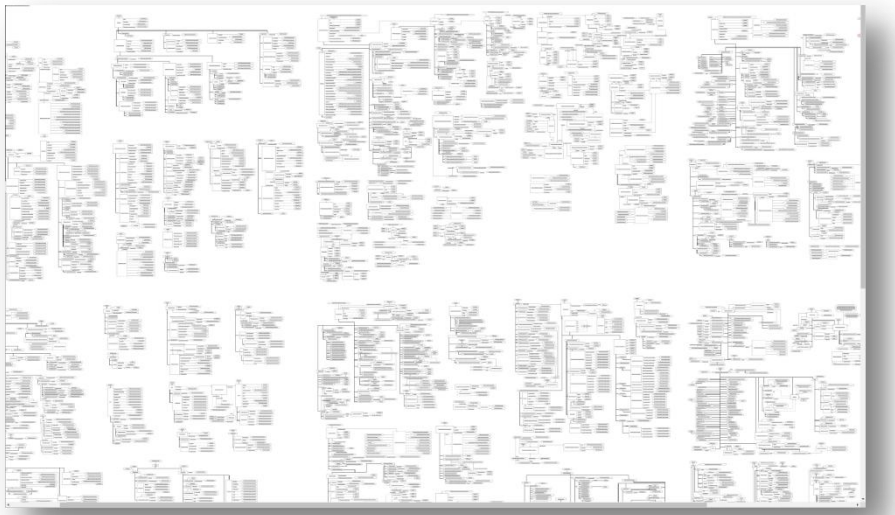
早期的国家级标准如法国的SET、德国的VDAFS和美国的IGES主要关注几何数据交换。随后，国际标准组织（ISO）推出了统一的国际标准——产品模型数据标准（STEP），以规范技术产品数据的各个方面。



All Rights Reserved

STEP应用程序协议（AP）

STEP应用程序协议（AP）定义了如何在不同的计算机系统和组织之间交换产品数据，确保了数据的一致性和准确性。以下是一系列STEP应用程序协议的概述，每个协议都专门针对特定的行业和设计需求：



STEP涉及到的各领域概念和关系图
https://www.steptools.com/stds/stp_expg/arm.html

AP 201：核心制图
AP 202：关联制图
AP 203：配置控制设计
AP 203e2：配置控制设计（第二版）
AP 209：复合材料和金属结构分析及相关设计
AP 210：电气设计
AP 214：汽车设计
AP 215：船舶安排
AP 216：船舶模塑
AP 218：船舶结构
AP 219：尺寸测量资源
AP 221：加工过程功能数据
AP 223：铸造
AP 224：基于特征的制造过程规划
AP 225：建筑构件
AP 227：工厂布局
AP 232：包装技术数据
AP 235：工程特性
AP 236：家具设计与室内设计
AP 238：STEP-NC 机床操作
AP 239：产品生命周期支持
AP 240：宏过程规划
AP 242：基于模型的3D工程

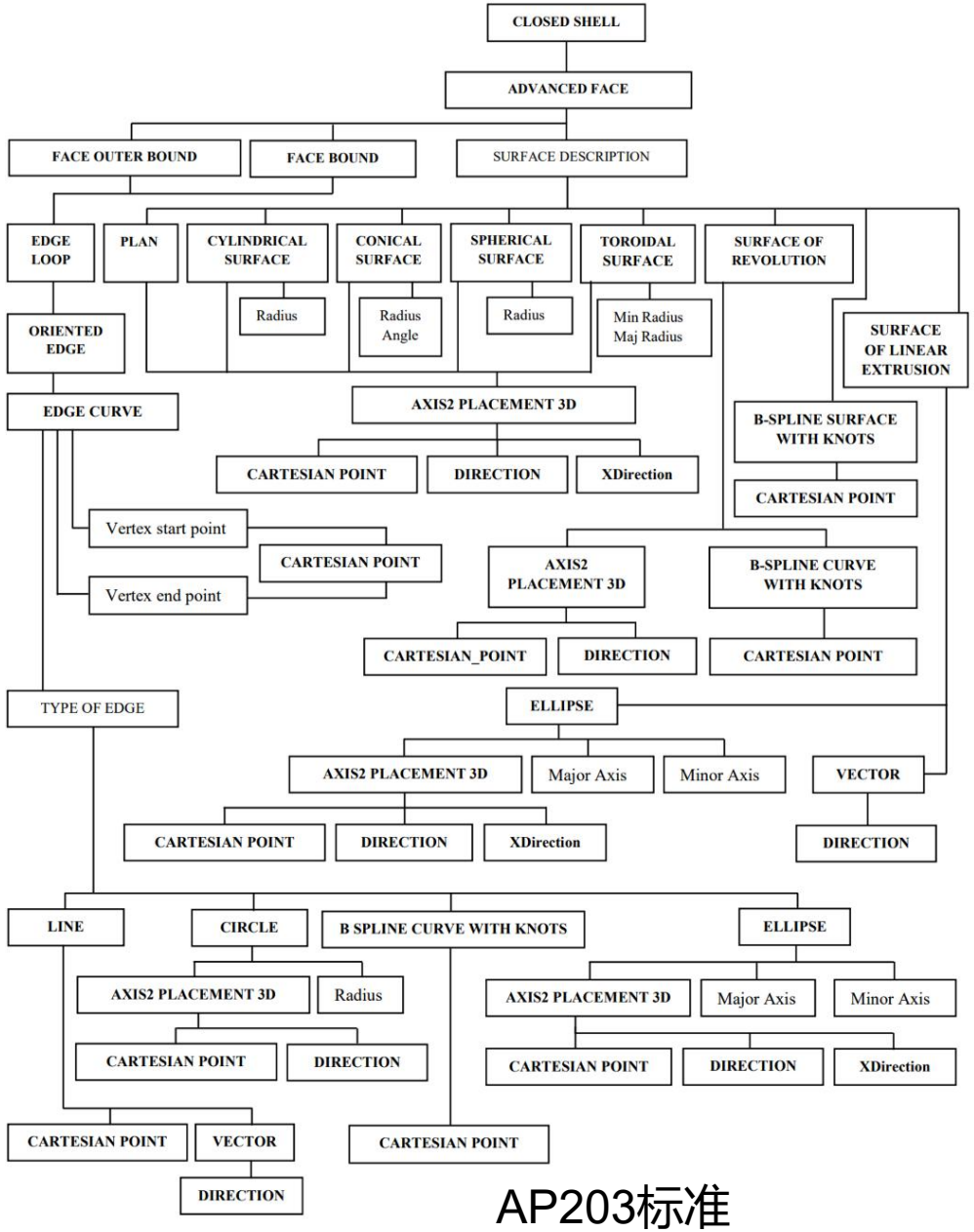
All Rights Reserved

AP203

AP203是最常用的标准，它描述了体、面、线、点的组织形式，即Brep的实现标准。

随着AP242等新标准的推出，AP203的一些功能已经被新的标准所取代。AP242结合了AP203和AP214的特点，并增加了更多的功能，如产品制造信息（PMI）和公差数据，以支持更高级的数据内容和应用。

尽管如此，AP203仍然是一个重要的工业标准，对于理解和使用STEP文件格式具有重要意义。



AP203标准

All Rights Reserved

STEP标准的表达- Express语言

STEP标准的表达实现采用了Express语言。Express是一种用于计算机辅助设计 (CAD) 的领域建模语言，用以定义和描述3D几何和拓扑信息，它是一种声明式、基于规则的语言。 Express语言基本语法遵循面向对象的原则，使用实体、属性和关系来描述3D几何和拓扑信息。语法主要分为实体类型定义、属性定义、实体关系定义和函数定义四个部分。

1. 实体类型定义：用于定义一个实体的几何和拓扑属性。一个实体类型由一个名称、一个或多个属性和一个或多个超类（基类）组成。例如：

```
ENTITY cylinder;
SUBTYPE OF (solid);
radius : positive_length_measure;
height : positive_length_measure;
END_ENTITY;
```

2. 属性定义：属性定义用于定义一个实体的某种特定属性，例如长度、面积、颜色等。属性可以是实数、整数、布尔值、字符串等类型。例如：

```
ATTRIBUTE radius : positive_length_measure;
```

All Rights Reserved

3.实体关系定义：
实体关系定义用于定义实体之间的关系，例如包含、属于、连接等。例如：

```
ENTITY line;
SUBTYPE OF (curve);
pnt : ARRAY [1:2] OF point;
DERIVE dir : vector := pnt[2] - pnt[1];
length : positive_length_measure :=
dir.magnitude;
END_ENTITY;
```

这段代码定义了一个实体类型名为line，它是curve的子类型。该实体类型包含了两个名为pnt的点（point），用一个大小为2的数组来存储。接着，它计算出这两个点形成的向量dir，并将其定义为该实体类型的派生属性（derived attribute）。length也是一个派生属性，它是向量dir的长度（magnitude），表示线段的长度。该实体类型没有其他的属性或关系，它仅仅表示了一条线段的两个端点和长度信息。

4. 函数定义：
函数定义用于定义一个或多个输入值，并返回一个或多个输出值。函数可以用于计算属性值、几何关系、逻辑运算等。例如：

```
FUNCTION distance (p1, p2 : point) : real;
LOCAL
dx, dy, dz : real;
BEGIN
dx := p2.x - p1.x;
dy := p2.y - p1.y;
dz := p2.z - p1.z;
RETURN (sqrt(dxdx + dydy + dz*dz));
END_FUNCTION;
```

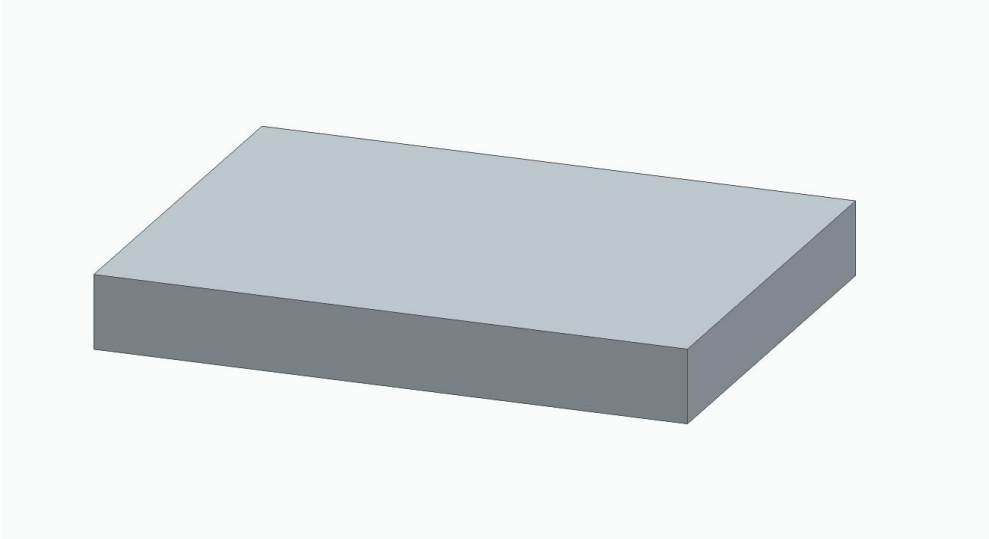
具体STEP中标准的实现和文件格式说明请参照网站https://www.steptools.com/ .

All Rights Reserved

一个STEP文件例子

一个盒子.step

```
...
#10 =
ADVANCED_BREP_SHAPE_REPRESENTATION('',(#11,#15),#
345);
#11 = AXIS2_PLACEMENT_3D('',#12,#13,#14);
#12 = CARTESIAN_POINT('',(0.,0.,0.));坐标系
#13 = DIRECTION('',(0.,0.,1.));
#14 = DIRECTION('',(1.,0.,-0.));
#15 = MANIFOLD_SOLID_BREP('',#16);
#16 =
CLOSED_SHELL(  ''  ,(#17,#137,#237,#284,#331,#338));
#17 = ADVANCED_FACE('',(#18),#32,.F.);
#18 = FACE_BOUND('',#19,.F.);
#19 = EDGE_LOOP('',(#20,#55,#83,#111));
#20 = ORIENTED_EDGE('',*,*,#21,.F.);
#21 = EDGE_CURVE('',#22,#24,#26,.T.);
#22 = VERTEX_POINT('',#23);
#23 = CARTESIAN_POINT('',(-40.,-30.,-5.));
#24 = VERTEX_POINT('',#25);
#25 = CARTESIAN_POINT('',(-40.,-30.,5.));
...
```



CLOSED_SHELL > 定义了3D模型的外表面
ADVANCED_FACE > 由边界围城的高级表面
FACE_BOUND > 边界包含了一个边界循环
EDGE_LOOP > 边界循环中，包含了4个边
ORIENTED_EDGE > 定义了一个有向边
EDGE_CURVE > 定义了边缘曲线
VERTEX_POINT > 定义了一个顶点
CARTESIAN_POINT > 定义了一个笛卡尔坐标

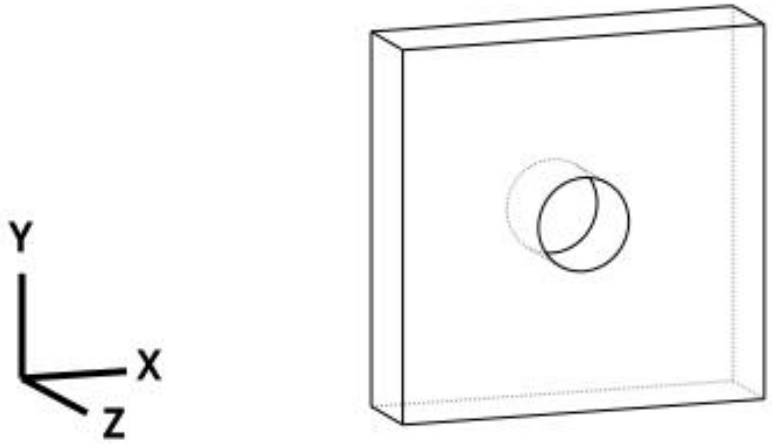
All Rights Reserved

3.几何建模工具CadQuery

什么是CadQuery?

CadQuery 是一个用于构建参数化 3D CAD 模型的 Python 库。它有几个特点：

- 使用尽可能接近您向人类描述对象的方式的脚本构建模型，使用标准的、已经建立的编程语言创建最终用户可以轻松自定义的参数化模型
- 除了传统的 STL 外，还可以输出高质量的 CAD 格式，如 STEP
- CadQuery基于OCP，是一组用于开源 OpenCasCade 建模内核的 Python 绑定。
- 使用 CadQuery，您可以使用非常少量的代码构建完全参数化的模型。




```
thickness = 0.5
width = 2.0
result= Workplane("front").box(width,
width,
thickness).faces(">Z").hole(thickness)
```

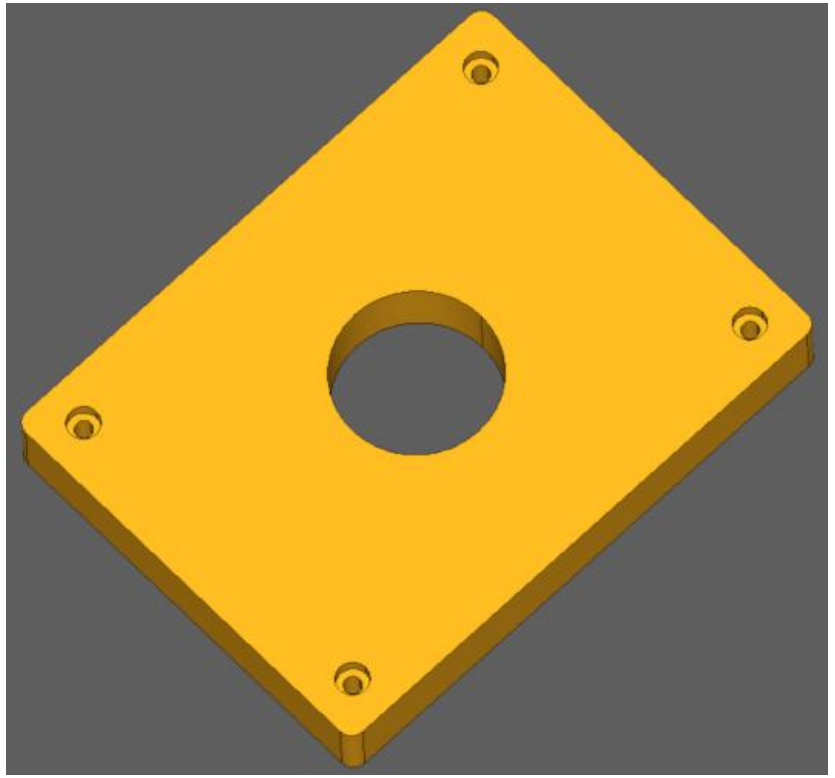
这个简单的脚本生成一个中间有孔的平板

All Rights Reserved

一个简单的例子：轴承托盘

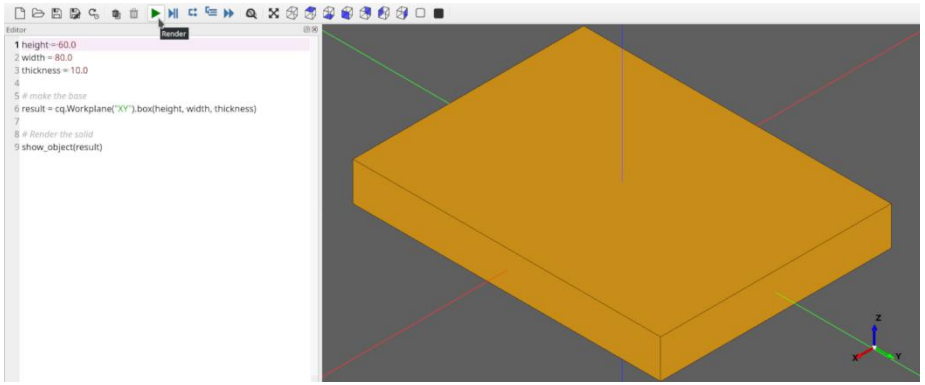
需求描述：
“一个 80 毫米 x 60 毫米 x 10 毫米的矩形块，在拐角处有用于安装 M2 内六角螺钉的沉头孔，在中间有一个直径为 22 毫米的圆形凹槽，用于安装轴承，矩形块的四周没有棱角”

需求分析:
矩形块 80*60*10
四个拐角处M2沉头孔
中心22毫米凹槽
四周倒角

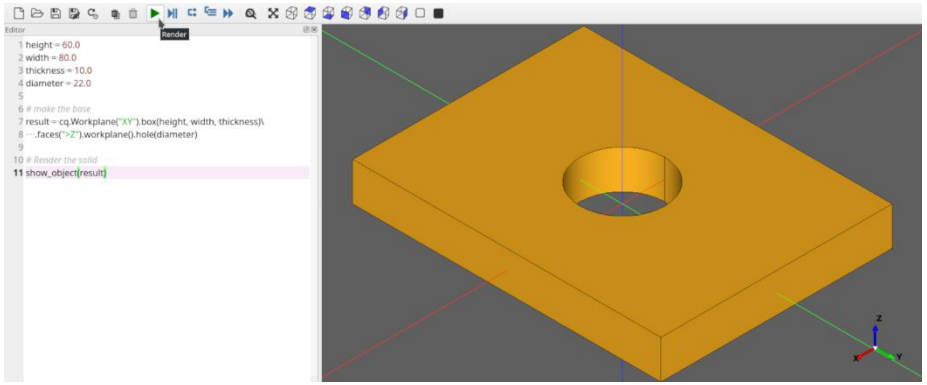


All Rights Reserved

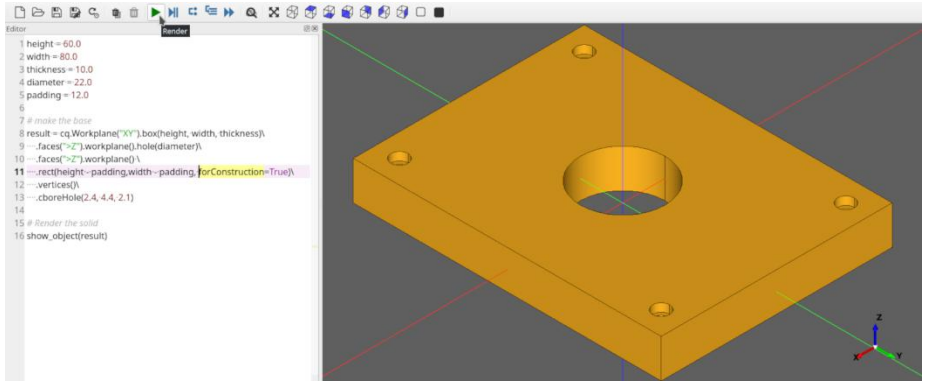
构建过程



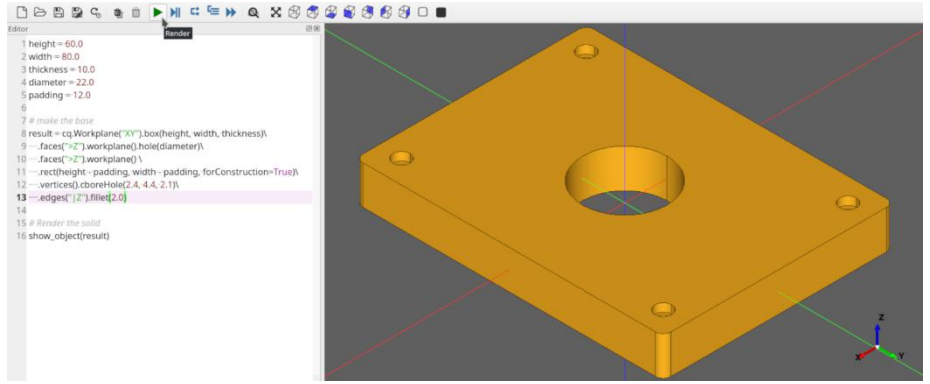
1 创建一个板



2 中心打孔



3 通过辅助线创建周边4个螺丝孔



4 将四周棱进行倒角

All Rights Reserved

代码和总结

设计原则

- 参数化的需求描述可以让代码变得有逻辑和复用能力
- 设计好工作平面和操作顺序，按照逻辑建立cq的执行堆栈
- 根据需求导出相应的文件格式给下游应用

这样就可以形成一个输入、输出明确，有逻辑的设计模块。这种模块化的思维和程序积累，可以帮助同学们处理更复杂庞大的系统。

```
# 定义参数
height = 60.0
 width = 80.0
 thickness = 10.0
 diameter = 22.0
 padding = 12.0

 # 创建执行栈
 result = (
     cq.Workplane("XY")
     .box(height, width, thickness)
     .faces(">Z")
     .workplane()
     .hole(diameter)
     .faces(">Z")
     .workplane()
     .rect(height - padding, width - padding, forConstruction=True)
     .vertices()
     .cboreHole(2.4, 4.4, 2.1)
     .edges("|Z")
     .fillet(2.0)
 )


 # 导出成不同格式的文件
 cq.exporters.export(result, "result.stl")
 cq.exporters.export(result.section(), "result.dxf")
 cq.exporters.export(result, "result.step")
```

All Rights Reserved

CadQuery基本概念

概念提纲：

3D BREP 拓扑概念
Workplanes 工作平面
2D构造草图
3D构造
Selectors 选择器
Construction Geometry 构造几何
The Stack 堆栈
Chaining 链式
The Context Solid 上下文实体
Iteration 迭代
Assemblies 装配体
约束求解
多层次API（Fluent API， Direct API， OCCT API）

All Rights Reserved

3D BREP 实现

在讨论 CadQuery 之前，有必要先讨论一下 3D CAD 拓扑。CadQuery 基于 OpenCascade 内核，该内核对对象使用边界表示 BREP。这就意味着，对象是由其封闭表面定义的。

顶点	空间中的一个点
边缘	沿特定路径（称为曲线）的连接两个或多个顶点
线	连接在一起的边的集合
面	一组边或线封闭而成
外壳	沿边缘连接在一起的面的集合
实心体	具有封闭内部的外壳
复合体	一组实心体的集合

All Rights Reserved

Workplanes 工作平面

大多数 CAD 程序都使用工作平面的概念。工作平面代表空间中的一个平面，从这个平面可以定位其他特征。

工作平面有一个中心点和一个本地坐标系，大多数创建对象的方法都是相对于当前工作平面创建对象的。通常创建的第一个工作平面是 "XY "平面，也称为 "front"平面。定义好实体后，创建工作平面最常用的方法是在实体上选择一个要修改的面，然后相对于该面创建一个新的工作平面。

您也可以在世界坐标系的任何位置创建新的工作平面，或使用偏移或旋转来创建相对于其他平面的工作平面。

工作平面最强大的功能是允许你在工作平面坐标系的二维空间中工作，然后 CadQuery 会将这些点从工作平面坐标系转换到世界坐标系，这样你的三维特征就会位于你想要的位置。这使得脚本的创建和维护变得更加容易。

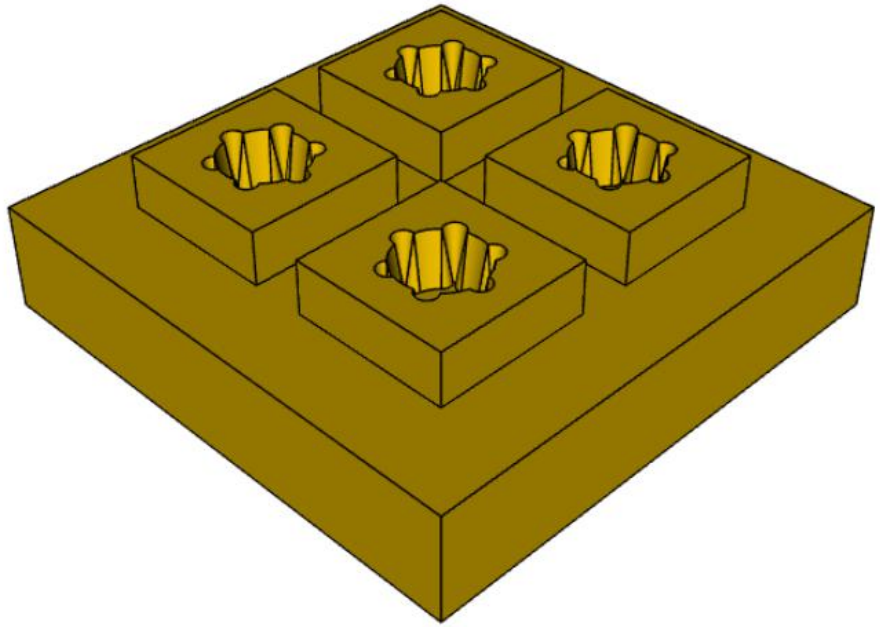
Workplane Class 工作平面类

Workplane 类包含当前选定的对象（形状列表、在objects属性中的向量或位置）、建模上下文（在ctx属性中）以及 CadQuery 的 api 方法。它是用户将实例化的主类。

All Rights Reserved

2D 通过草图构建

```
result = (
    cq.Workplane()
    .box(5, 5, 1)
    .faces(">Z")
    .workplane()
    .rarray(2, 2, 2, 2) # 生成一个矩形位置阵列
    .rect(1.5, 1.5)
    .extrude(0.5)
    .faces(">Z")
    .sketch()
    .circle(0.4)
    .wires() # 选定轮廓边缘
    .distribute(6) # 沿选定的边缘均匀分布
    .circle(0.1, mode="a") # a表示追加模式保留之前的几何体
    .clean()
    .finalize() # 完成草图绘制并返回父级
    .cutBlind(-0.5, taper=10)  # 切割深度和倒角大小
)
```



All Rights Reserved

3D直接构造

您可以直接构建3D原型，如方框、楔形、圆柱和球体。您还可以扫掠、挤压和展平 2D 几何图形以形成 3D 特征。当然基本的原始操作也是可用的。

Sweep（扫掠）：

1. **定义：** 沿着路径的方向创建一个截面的过程。可以想象成在一个轨迹上滑动一个截面，形成一个扫掠体。
2. **操作：** 选择一个截面（形状），然后定义一个路径，软件会沿着路径创建一个与截面相切的三维形体。

Extrude（挤压）：

1. **定义：** 沿着指定方向拉伸或压缩一个平面形状，以创建一个具有一定深度的三维对象。
2. **操作：** 选择一个平面形状，然后指定拉伸或压缩的方向和距离，软件将生成一个沿该方向伸展的三维体。

Loft（放样）：

1. **定义：** 使用两个或多个截面之间的过渡形成一个平滑的曲面或体积。
2. **操作：** 选择两个或多个截面，软件会在它们之间创建一个平滑的过渡形状，可以是曲面或实体。

All Rights Reserved

Selectors 选择器

选择器允许您选择一个或多个特征，以便定义新特征。例如，你可以挤出一个盒子，然后选择顶面作为新特征的位置。或者，你可以挤出一个盒子，然后选择所有的垂直边缘，这样就可以对它们应用圆角。
您可以使用选择器选择 Vertices顶点、Edges边、Faces面、Solids实体和 Wires线。
如果您使用传统 CAD 系统构建一个对象，选择器就相当于您的手和鼠标。

Construction Geometry 构造几何（辅助线）

构造几何体是不属于对象的一部分的特征，其定义只是为了帮助构建对象。一个常见的例子可能是定义一个矩形，然后用矩形的角来定义一组孔的位置。
大多数 CadQuery 构造方法都提供一个forConstruction关键字，该关键字创建的特征仅用于定位其他特征。

All Rights Reserved

The Stack 堆栈

当您使用 CadQuery 时，每个操作的结果都会返回一个新的 Workplane 对象。每个工作平面对象都有一个对象列表以及对其父对象的引用。

您可以通过从堆栈中移除当前对象，返回到以前的操作。例如
cq.Workplane(someObject).faces(">Z").first().vertices()
返回一个 CadQuery 对象，其中包含 someObject 对象最高面上的所有顶点。

但您始终可以在堆栈中向后移动以获取面：
cq.Workplane(someObject).faces(">Z").first().vertices().end()

All Rights Reserved

Chaining 链式

所有 Workplane 方法都会返回另一个 Workplane 对象，以便您可以将这些方法流畅地链接在一起。使用核心工作平面方法来获取创建的对象。
在这些链式调用过程中，每次生成一个新的 Workplane 对象时，它都有一个 parent 属性指向创建它的 Workplane 对象。多个 CadQuery 方法搜索此父链，例如在搜索上下文实体时。您还可以给Workplane 对象一个标签，并且在调用链的下游，您可以使用标签引用该特定对象。

```
# 定义一个工作平面并设置标签
result = cq.Workplane("XY").tag("my_workplane")
# 在工作平面上创建一个长方体
result = result.box(1, 1, 1)
# 定义另一个工作平面并设置标签
result = result.faces(">Z").workplane(offset=2).tag("another_workplane")
# 在另一个工作平面上创建一个圆柱体
result = result.circle(0.5).extrude(1)
# 切换回之前定义的工作平面并在其上创建一个球体
result = result.workplaneFromTagged("my_workplane").sphere(0.75)
# 显示结果
show_object(result)
```

workplaneFromTagged切换回之前定义的工作平面

All Rights Reserved

The Context Solid 上下文实体

大多数情况下，您都是在创建一个单一对象，并为该单一对象添加特征。CadQuery 会观察你的操作，并将创建的第一个实体对象定义为 "上下文实体"。之后，您创建的任何特征都会自动与该实体相结合（除非您另行指定）。即使实体是在堆栈的较远处创建的，也会发生这种情况。
例如：

```
cq.Workplane("XY").box(1, 2, 3).faces(">Z").circle(0.25).extrude(1)
```

将创建一个 1x2x3 的盒子，顶面延伸出一个圆柱形凸台。无需手动组合通过长方体挤压出来的圆，因为挤出的默认行为是将结果与上下文实体结合起来。

hole() 方法的工作原理类似，如果您想避免这种情况，可以指定combine=False，CadQuery 将单独创建实体。

```
cq.Workplane("XY").box(1, 2, 3).faces(">Z").circle(0.25).extrude(1,combine = True)
```

Iteration 迭代

CAD 模型经常有重复的几何图形，采用 for 循环来构建特征确实很烦人。许多 CadQuery 方法会自动对堆栈中的每个元素进行操作，因此您不必编写循环。例如：

```
cq.Workplane("XY").box(1, 2, 3).faces(">Z").vertices().circle(0.5)
```

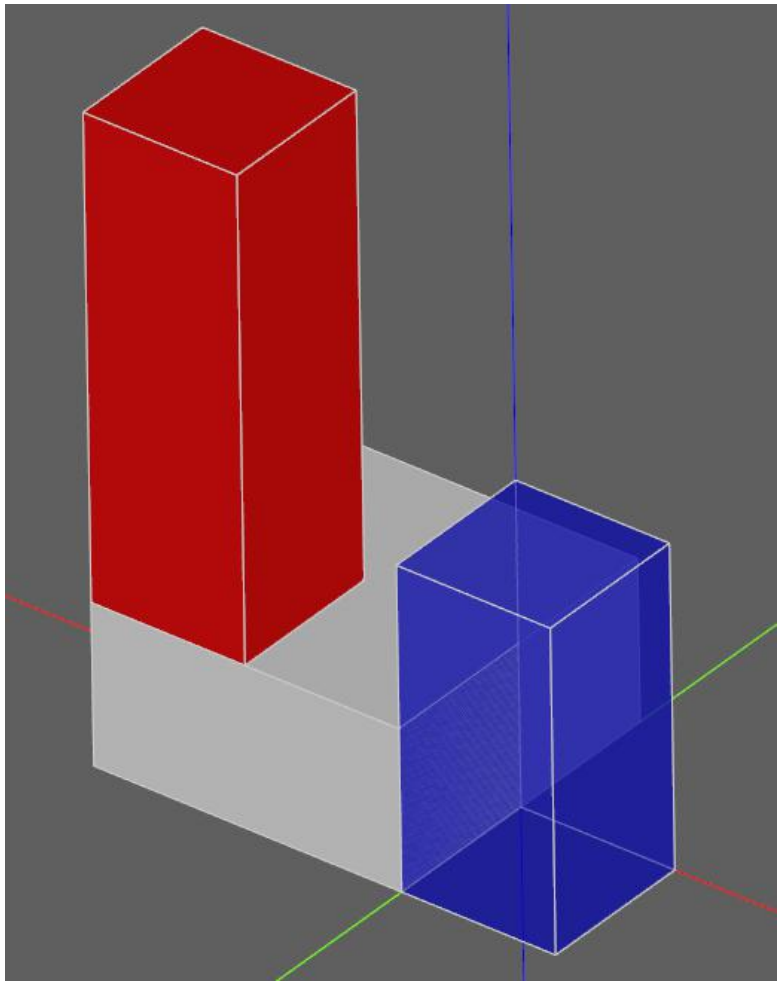
实际上会创建 4 个圆，因为vertices()选择了矩形面的 4 个顶点，circle()方法会遍历堆栈中的每个成员。

All Rights Reserved

Assemblies 装配体

装配体是由简单的模型经过组合，形成复杂的、可能嵌套的组件。

```
w = 10
d = 10
h = 10

part1 = Workplane().box(2 * w, 2 * d, h)
part2 = Workplane().box(w, d, 2 * h)
part3 = Workplane().box(w, d, 3 * h)

assy = (
    Assembly(part1, loc=Location(Vector(-w, 0, h / 2))).add(part2,
loc=Location(Vector(1.5 * w, -0.5 * d, h / 2)), color=Color(0, 0, 1, 0.5)).add(part3,
loc=Location(Vector(-0.5 * w, -0.5 * d, 2 * h)), color=Color("red"))
)
```



All Rights Reserved

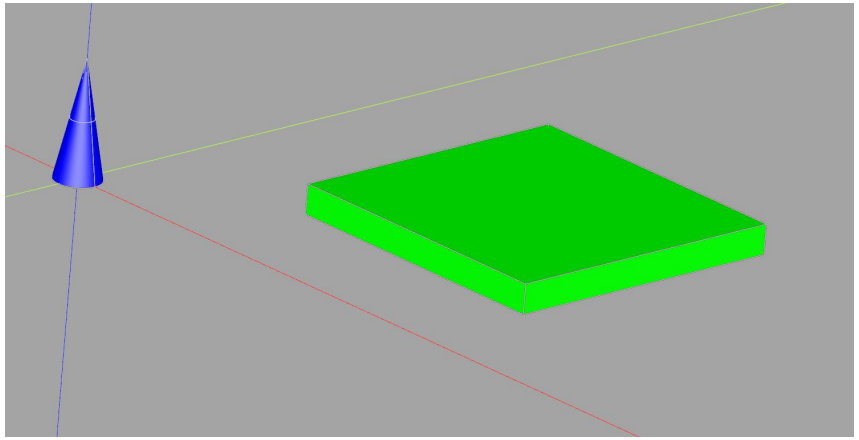
约束求解

轴约束：两个法向量反重合或它们之间的角度（以弧度为单位）等于指定值。可以为所有具有一致法线矢量的实体定义 - 平面、线和边。

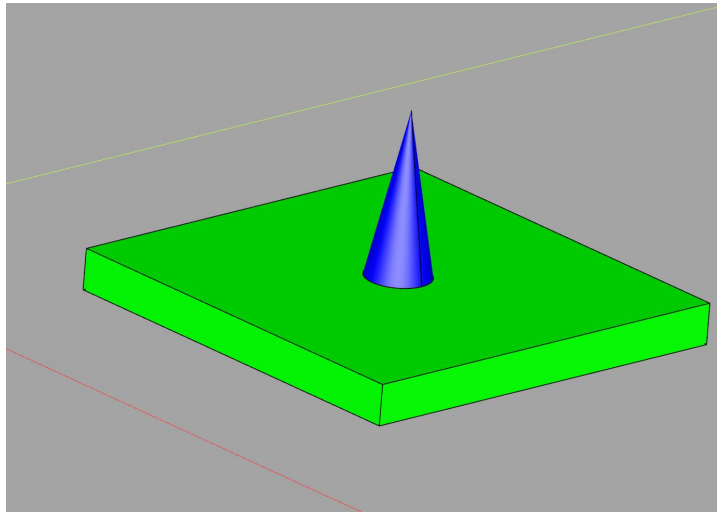
点约束：两个点重合或相距指定距离。可以为所有实体定义，质心用于线、面、实体，顶点位置用于顶点。

平面约束 : Axis: 和 :Point: 约束的组合。

```
box = cq.Workplane().center(10,10).box(10, 10, 1)
cone = cq.Solid.makeCone(0.8, 0, 4)

assy = cq.Assembly()
assy.add(box, name="box", color=cq.Color("green"))
assy.add(cone, name="cone", color=cq.Color("blue"))
# place the center of the flat face of the cone in the center of the upper face of the plate
assy.constrain(" box@faces@>Z", "cone@faces@<Z", "Point")

# set both the flat face of the cone and the upper face of the plate to point in the same direction
assy.constrain(" box@faces@<Z", "cone@faces@<Z", "Axis", param=0)

# assy.solve()  # 约束求解
show_object(assy)
```



约束求解前



约束求解后

All Rights Reserved

The Fluent API

我们所说的 Fluent API 就是您第一次开始使用 CadQuery 时所使用的，该类Workplane及其所有方法定义了 Fluent API。这是您在大多数情况下都会使用和看到的 API，它相当容易使用，并且为您简化了很多事情。一个经典的例子是：

```
part = cq.Workplane("XY").box(1, 2,3).faces(">Z").vertices().circle(0.5).cutThruAll()
```

在这里，我们创建一个Workplane对象，随后在该对象上调用多个方法来创建我们的零件。一般 Fluent API 将Workplane视为部件对象，并将其所有方法视为影响部件的操作。通常您会从一个空的 Workplane 开始，然后通过调用 Workplane 的方法添加更多特征。
在 CadQuery 代码中使用的传统代码风格，可以很好地看出修改零件操作的这种分层结构。使用 CadQuery Fluent API 编写的代码通常如下所示：

```
part = cq.Workplane("XY").box(1, 2, 3).faces(">Z").vertices().circle(0.5).cutThruAll()
```

或者像这样：

```
part = Workplane("XY") part = part.box(1, 2, 3) part = part.faces(">Z") part = part.vertices() part = part.circle(0.5) part = part.cutThruAll()
```

All Rights Reserved

The Direct API

Direct API 是被 Fluent API 调用的底层API。9 个拓扑类及其方法组成了 Direct API。这些类实际上封装了 Open CASCADE Technology (OCCT) 类。
9 个拓扑类别是：
1.Shape
2.Compound
3.CompSolid
4.Solid
5.Shell
6.Face
7.Wire
8.Edge
9.Vertex

例如:

```
circle_wire = cq.Wire.makeCircle(10, cq.Vector(0, 0, 0), cq.Vector(0, 0, 1))
circular_face = cq.Face.makeFromWires(circle_wire, [])
```

All Rights Reserved

The OCCT API

OCCT API 是 CadQuery 的最低层。Direct API 构建于 OCCT API 之上，其中 CadQuery 中的 OCCT API 通过 OCP 提供。OCP 是 CadQuery 使用的 OCCT C++ 库的 Python 绑定。这意味着您可以访问 Python 和 CadQuery 中的（几乎）所有 OCCT C++ 库。使用 OCCT API 将为您提供更大的灵活性和控制力，但它非常繁琐且难以使用。

要获得这些知识，最有效的方法是：
1.阅读 Direct API 源代码，因为它是基于 OCCT API 构建的，有很多用法示例。
2.浏览OCCT的C++ 文档

导入 OCCT API 的特定类的一般方法是
```
from OCP.thePackageName import theClassName
```

例如，如果您想使用 BRepPrimAPI_MakeBox 类。您将通过以下方式
```
from OCP.BRepPrimAPI import BRepPrimAPI_MakeBox
```

任何类的包名称都写在文档页面的顶部

All Rights Reserved

Fluent API <=> Direct API

以下是您从 Direct API 获取对象（即拓扑对象）的所有可能性。
您可以结束 Fluent API 调用链并使用 Workplane.val() 获取堆栈上的最后一个对象，或者您也可以使用 Workplane.vals() 获取所有对象

```
>>> box = cq.Workplane().box(10, 5, 5)
>>> print(type(box))
<class cadquery.cq.Workplane>
>>> box = cq.Workplane().box(10, 5, 5).val()
>>> print(type(box))
<class cadquery.occ_impl.shapes.Solid>
```

如果您只想获取 Workplane 的上下文实体，您可以使用 Workplane.findSolid()：

```
>>> part = cq.Workplane().box(10,5,5).circle(3).val()
>>> print(type(part))
<class cadquery.cq.Wire>
>>> part = cq.Workplane().box(10,5,5).circle(3).findSolid()
>>> print(type(part))
<class cadquery.occ_impl.shapes.Compound> # findSolid 的返回类型是实体或复合对象
```

如果您想反其道而行之，即在 Fluent API 中使用拓扑 API 中的对象，
您可以选择：您可以将拓扑对象作为基础对象传递给该Workplane对象。

```
solid_box = cq.Solid.makeBox(10, 10, 10)
part = cq.Workplane(obj=solid_box) # 您可以继续使用 Fluent API 建模
part = part.faces(">Z").circle(1).extrude(10)
```

您可以在 Fluent API 调用链中，用 Workplane.newObject() 添加一个拓扑对象作为一个新的操作或步骤：

```
circle_wire = cq.Wire.makeCircle(1, cq.Vector(0, 0, 0), cq.Vector(0, 0, 1))
box = cq.Workplane().box(10, 10, 10).newObject([circle_wire]) # 继续构建模型
box = ( box.toPending().cutThruAll() ) # 注意，如果要在后续操作中使用它，需要调用 `toPending`
```

All Rights Reserved

Direct API <=> OCCT API

Direct API 的每个对象，都在 wrapped 属性中存储其对应的 OCCT 对象：

```
>>> box = cq.Solid.makeBox(10,5,5)
>>> print(type(box))
<class cadquery.occ_impl.shapes.Solid>
>>> box = cq.Solid.makeBox(10,5,5).wrapped
>>> print(type(box))
<class OCP.TopoDS.TopoDS_Solid>
```

如果您想将 OCCT 对象转换为 Direct API 对象，只需将其作为目标类的参数传递即可：

```
>>> from OCP.BRepPrimAPI import BRepPrimAPI_MakeBox
>>> occt_box = BRepPrimAPI_MakeBox(5,5,5).Solid()
>>> print(type(occt_box))
<class OCP.TopoDS.TopoDS_Solid>
>>> direct_api_box = cq.Solid(occt_box)
>>> print(type(direct_api_box))
<class cadquery.occ_impl.shapes.Solid>
```

All Rights Reserved

4. 上机实例

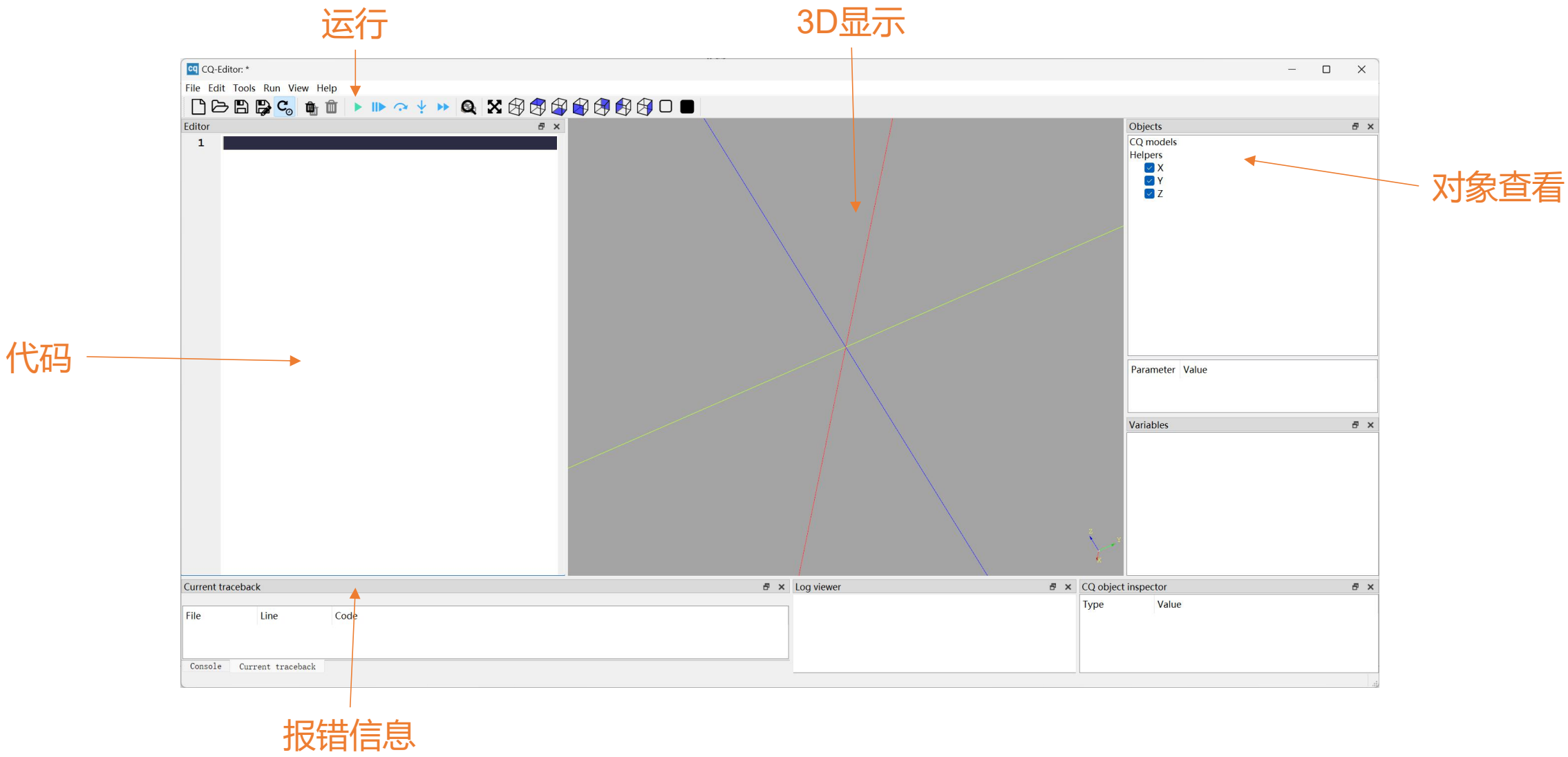
上机环境

使用MSCAX环境或自建开发环境

- MSCAX环境（支持Ubuntu22.04，5.5节详细介绍）
- 自建官方环境（支持常见windows、Linux等）
  - 安装编程语言: Python>=3.8
  - 安装cq库: pip install cadquery
  - 安装cq官方开发环境cq-editor

All Rights Reserved

Cq-editor

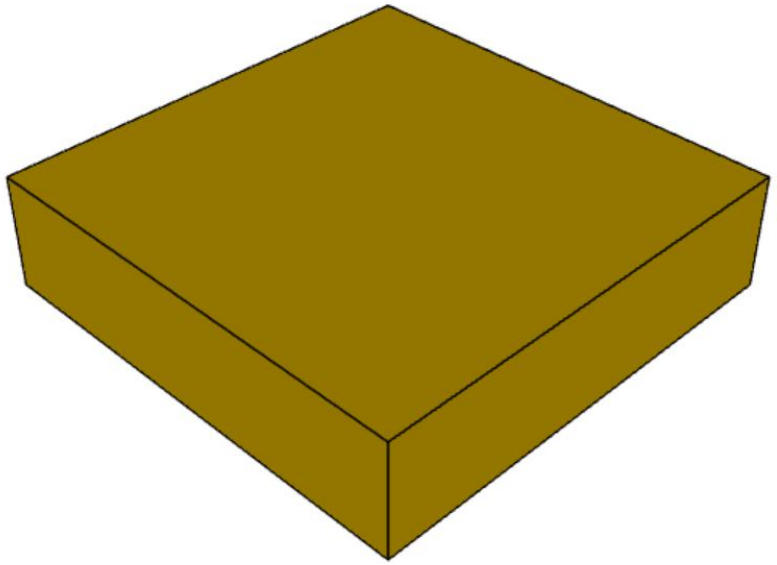


All Rights Reserved

1 构建盒子

最简单的例子，一个长方形的盒子

```
result = cq.Workplane("front").box(2.0, 2.0, 0.5)
```



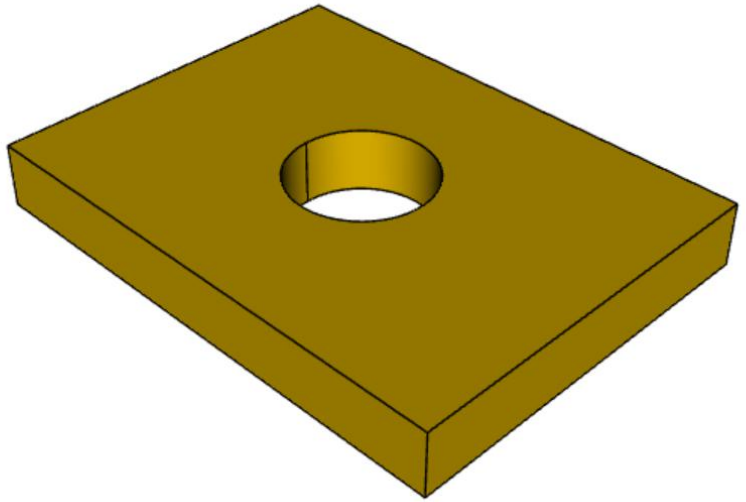
Workplane() .box()

All Rights Reserved

2 打孔

“>Z”选择结果框的最顶端。
个工作平面的原点位于 （0,0,0） 投影位于面的中心。
默认孔深度贯穿整个零件

```
# The dimensions of the box. These can be modified rather than changing the
# object's code directly.
length = 80.0
height = 60.0
thickness = 10.0
center_hole_dia = 22.0

# Create a box based on the dimensions above and add a 22mm center hole
result = (
    cq.Workplane("XY")
    .box(length, height, thickness)
    .faces(">Z")
    .workplane()
    .hole(center_hole_dia)
)
```



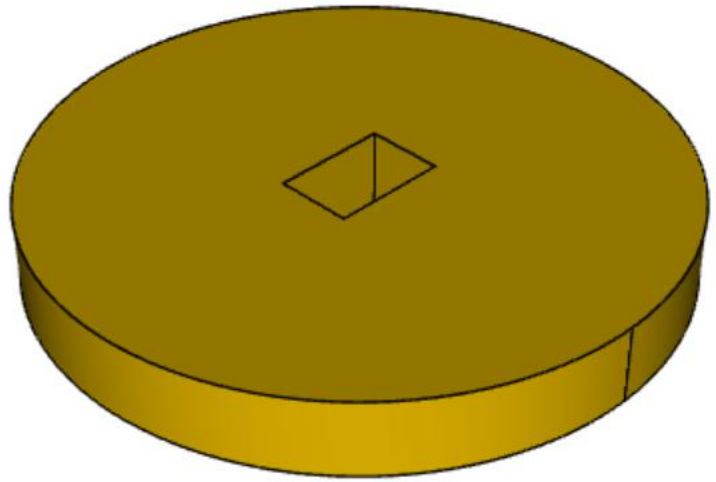
Workplane.hole()

All Rights Reserved

3 挤压操作

使用挤压构建棱柱形实体

```
result = cq.Workplane("front").circle(2.0).rect(0.5,
0.75).extrude(0.5)
```



Workplane.circle()

Workplane.rect()

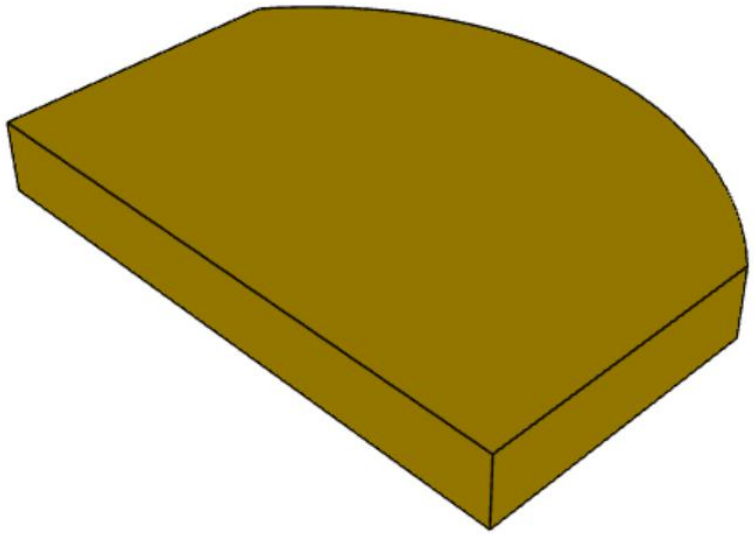
Workplane.extrude()

All Rights Reserved

4 曲线

2D 操作维护当前点，
最初位于原点，
使用 close（） 设置闭合曲线。

```
result = (
    cq.Workplane("front")
    .lineTo(2.0, 0)
    .lineTo(2.0, 1.0)
    .threePointArc((1.0, 1.5), (0.0, 1.0))
    .close()
    .extrude(0.25)
)
```



Workplane.lineTo()
Workplane.close()
Workplane.threePointArc()

All Rights Reserved

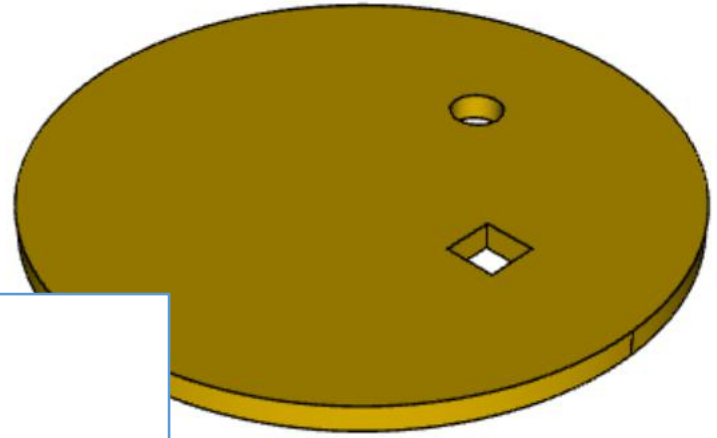
5 移动工作中心点

在此示例中，需要一个封闭的轮廓，以及一些内部特征。

此示例还演示了使用多行代码而不是较长的链式命令，当然在这种情况下，也可以在一长行中执行此操作。

可以在任何时候建立新的工作飞机中心。

```
result = cq.Workplane("front").circle(
    3.0
)  # current point is the center of the circle, at (0, 0)
result = result.center(1.5, 0.0).rect(0.5, 0.5)  # new work center is (1.5, 0.0)

result = result.center(-1.5, 1.5).circle(0.25)  # new work center is (0.0, 1.5).
# The new center is specified relative to the previous center, not global coordinates!

result = result.extrude(0.25)
```



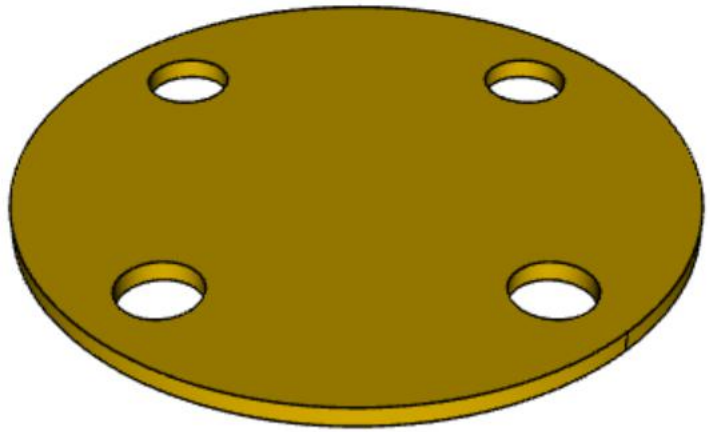
Workplane.center()

All Rights Reserved

6 点列表

您可以使用点列表一次构造多个对象。大多数构造方法（如 Workplane.circle() 和 Workplane.rect() ）如果它们在堆栈上，则将在多个点上运行

```
r = cq.Workplane("front").circle(2.0)  # make base
r = r.pushPoints(
    [(1.5, 0), (0, 1.5), (-1.5, 0), (0, -1.5)]
)  # now four points are on the stack
r = r.circle(0.25)  # circle will operate on all four points
result = r.extrude(0.125)  # make prism
```



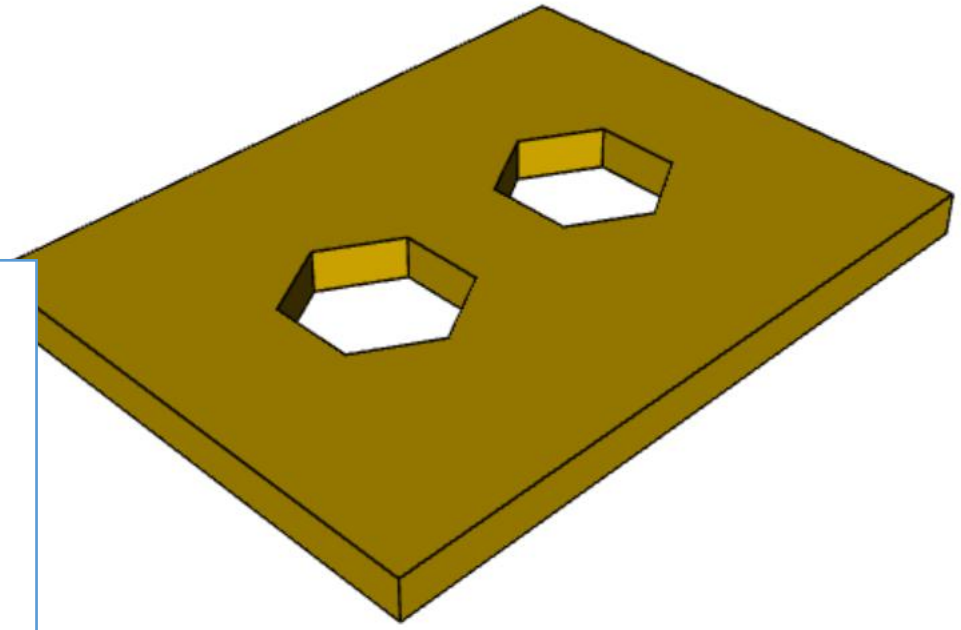
Workplane.pushPoints()

All Rights Reserved

7 多边形

可以为每个堆叠点创建多边形

```
result = (
    cq.Workplane("front")
    .box(3.0, 4.0, 0.25)
    .pushPoints([(0, 0.75), (0, -0.75)])
    .polygon(6, 1.0)
    .cutThruAll()
)
```



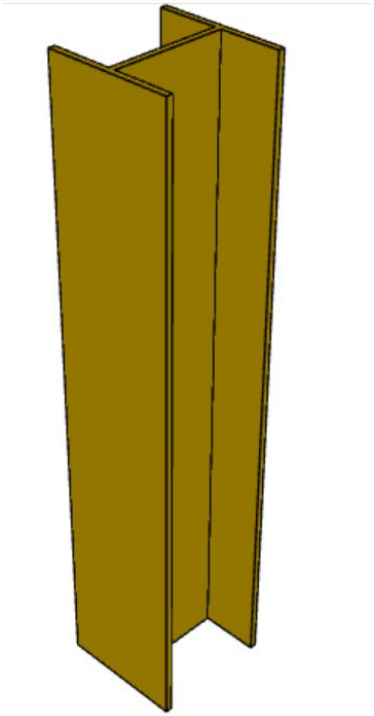
Workplane.polygon()

All Rights Reserved

8 折线

使用折线创建工字梁形状的一半，再通过镜像以创建最终轮廓。

```
(L, H, W, t) = (100.0, 20.0, 20.0, 1.0)
pts = [
    (0, H / 2.0),
    (W / 2.0, H / 2.0),
    (W / 2.0, (H / 2.0 - t)),
    (t / 2.0, (H / 2.0 - t)),
    (t / 2.0, (t - H / 2.0)),
    (W / 2.0, (t - H / 2.0)),
    (W / 2.0, H / -2.0),
    (0, H / -2.0),
]
result =
cq.Workplane("front").polyline(pts).mirrorY().extrude(L)
```



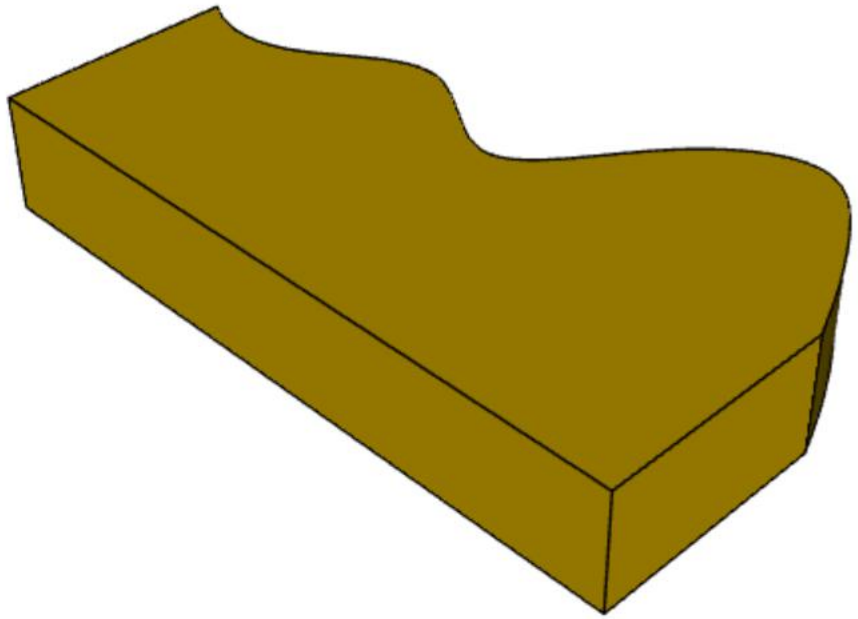
Workplane.polyline()

All Rights Reserved

9 样条

使用样条曲线通过点集合来定义边，可以创建复杂形状

```
s = cq.Workplane("XY")
sPnts = [
    (2.75, 1.5),
    (2.5, 1.75),
    (2.0, 1.5),
    (1.5, 1.0),
    (1.0, 1.25),
    (0.5, 1.0),
    (0, 1.0),
]
r = s.lineTo(3.0, 0).lineTo(3.0, 1.0).spline(sPnts,
includeCurrent=True).close()
result = r.extrude(0.5)
```

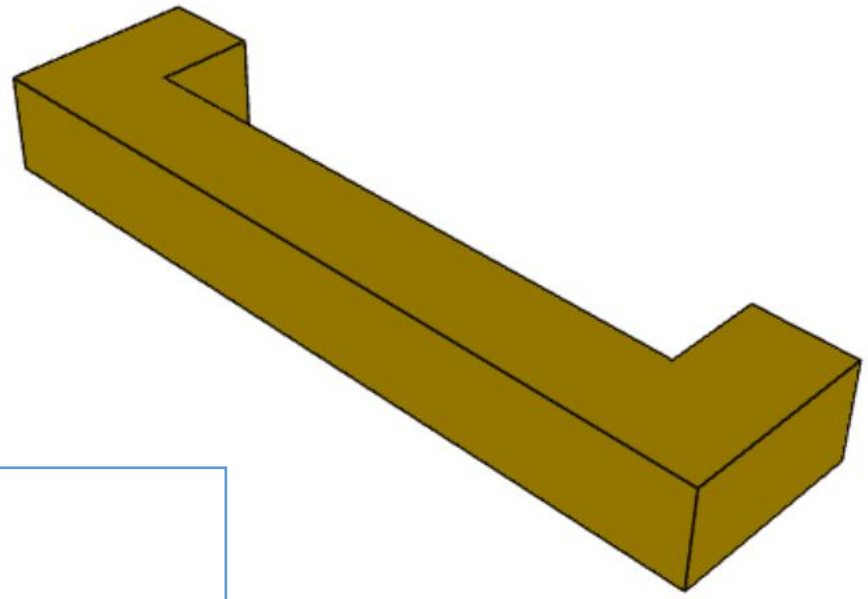


Workplane.spline()

All Rights Reserved

10 镜像（2D）

当形状是对称时，可以镜像 2D 几何体。
水平线和垂直线，帮助更好创建图形



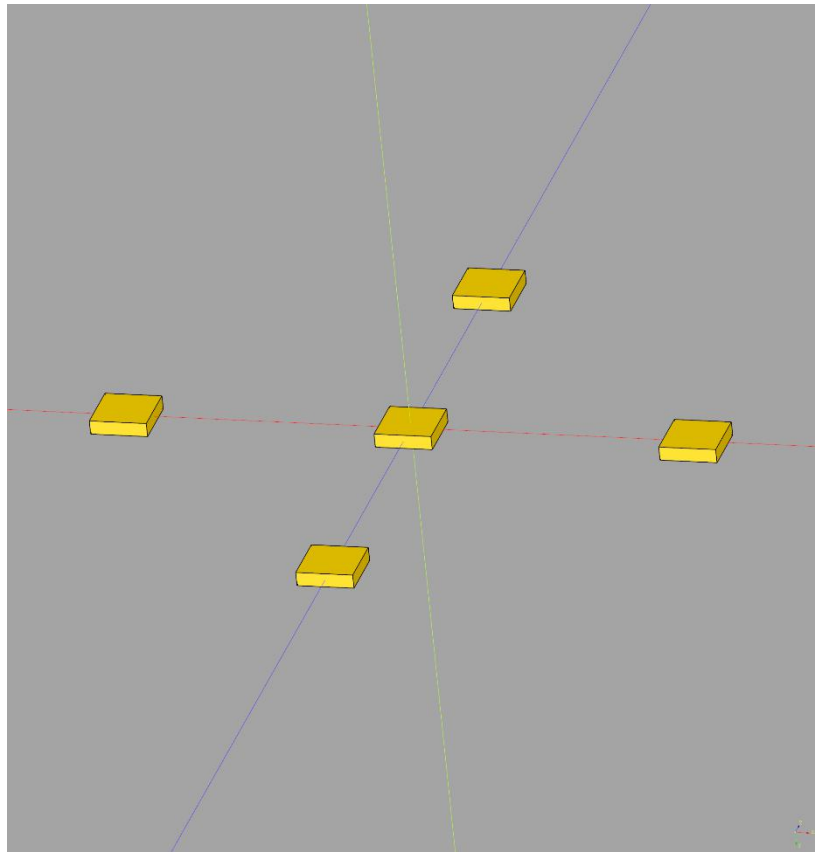
```
r = cq.Workplane("front").hLine(1.0)  # 1.0 is the distance, not coordinate
r = (
    r.vLine(0.5).hLine(-0.25).vLine(-0.25).hLineTo(0.0)
)  # hLineTo allows using xCoordinate not distance
result = r.mirrorY().extrude(0.25)  # mirror the geometry and extrude
```

Workplane.hLine()
Workplane.vLine()
Workplane.hLineTo()
Workplane.mirrorY()
Workplane.mirrorX()

All Rights Reserved

11 镜像（3D）

```
result = cq.Workplane("front").box(2.0, 2.0, 0.5)

result = result.rotate((0, 0, 0), (1, 0, 0), 90)

mirXY_neg = result.mirror(mirrorPlane="XY",
basePointVector=(0, 0, -5))
mirXY_pos = result.mirror(mirrorPlane="XY",
basePointVector=(0, 0, 5))
mirZY_neg = result.mirror(mirrorPlane="ZY",
basePointVector=(-5, 0, 0))
mirZY_pos = result.mirror(mirrorPlane="ZY",
basePointVector=(5, 0, 0))

result.union(mirXY_neg).union(mirXY_pos).union(mirZY_n
eg).union(mirZY_pos)
```

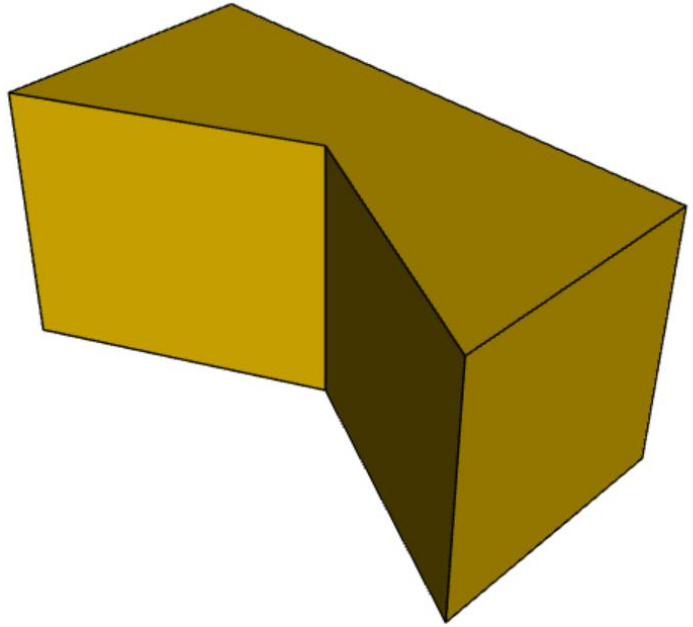


Workplane.mirror()
Workplane.union()
Workplane.rotate()

All Rights Reserved

12 镜像(平面)

```
result = cq.Workplane("XY").line(0, 1).line(1, 0).line(0, -0.5).close().extrude(1)

result = result.mirror(result.faces(">X"), union=True)
```

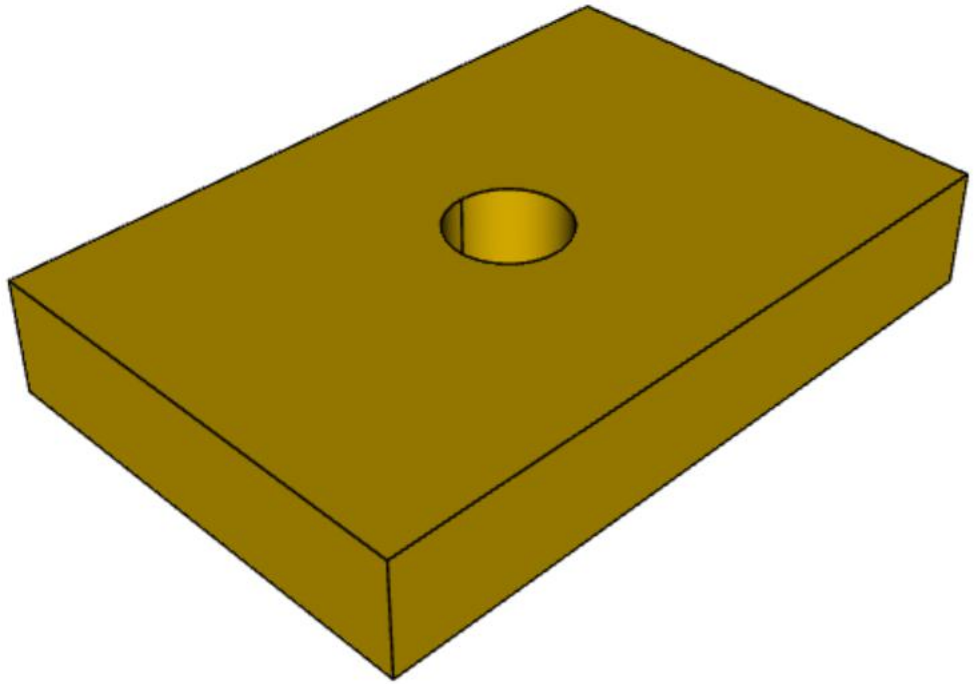


Workplane.faces()

All Rights Reserved

13 定义新的工作面（基于面）

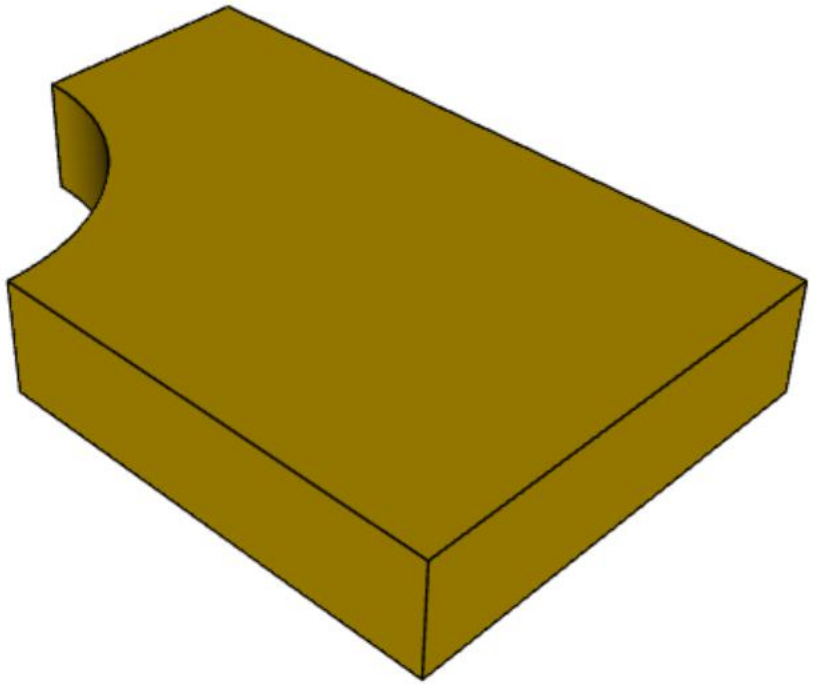
```
result = cq.Workplane("front").box(2, 3, 0.5)  # make a
basic prism
result = (
    result.faces(">Z").workplane().hole(0.5)
)  # find the top-most face and make a hole
```



All Rights Reserved

14 定义新的工作面（基于点）

```
result = cq.Workplane("front").box(3, 2, 0.5) result = (

result.faces(">Z").vertices("<XY").workplane(centerOption
="CenterOfMass")
)
result = result.circle(1.0).cutThruAll()
```

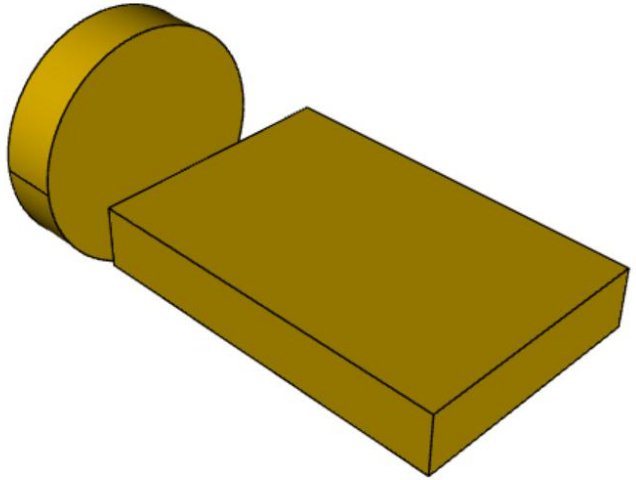


Workplane.vertices()
Workplane.cutThruAll()

All Rights Reserved

15 定义新的工作面（基于偏移）

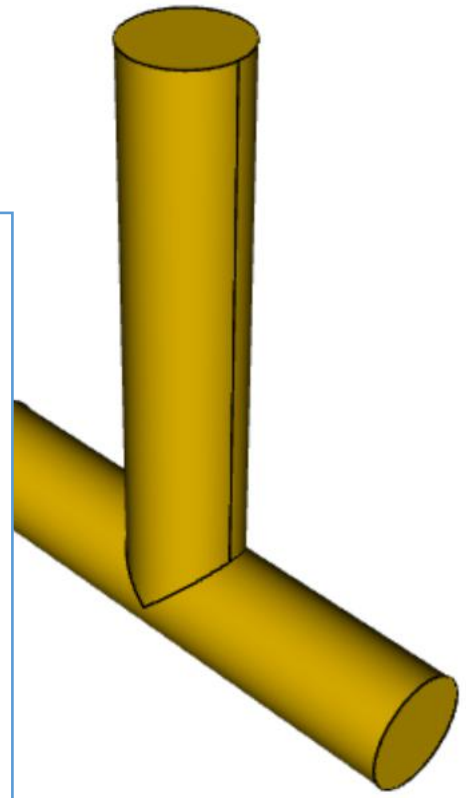
```
result = cq.Workplane("front").box(3, 2, 0.5)  # make a
basic prism
result = result.faces("<X").workplane(
    offset=0.75
)  # workplane is offset from the object surface
result = result.circle(1.0).extrude(0.5)  # disc
```



All Rights Reserved

16 复制工作面

```
result = (
    cq.Workplane("front")
    .circle(1)
    .extrude(10)  # make a cylinder
    # We want to make a second cylinder perpendicular to the first,
    # but we have no face to base the workplane off
    .copyWorkplane(
        # create a temporary object with the required workplane
        cq.Workplane("right", origin=(-5, 0, 0))
    )
    .circle(1)
    .extrude(10)
)
```



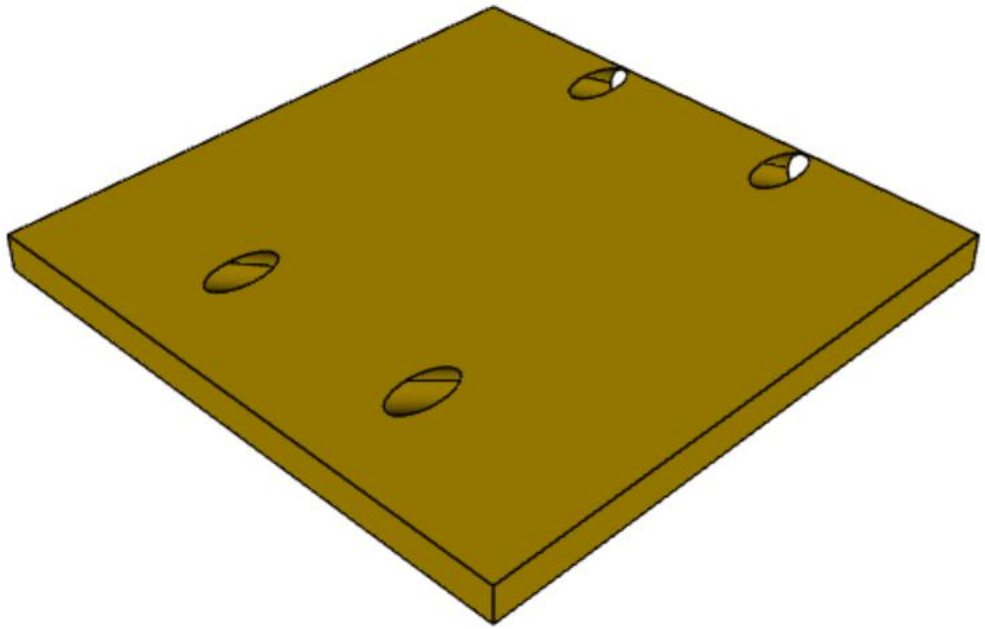
Workplane.copyWorkplane()

All Rights Reserved

17 旋转工作面

您可以通过指定相对于另一个工作平面的旋转角度来创建旋转工作平面

```
result = (
    cq.Workplane("front")
    .box(4.0, 4.0, 0.25)
    .faces(">Z")
    .workplane()
    .transformed(offset=cq.Vector(0, -1.5, 1.0),
rotate=cq.Vector(60, 0, 0))
    .rect(1.5, 1.5, forConstruction=True)
    .vertices()
    .hole(0.25)
)
```

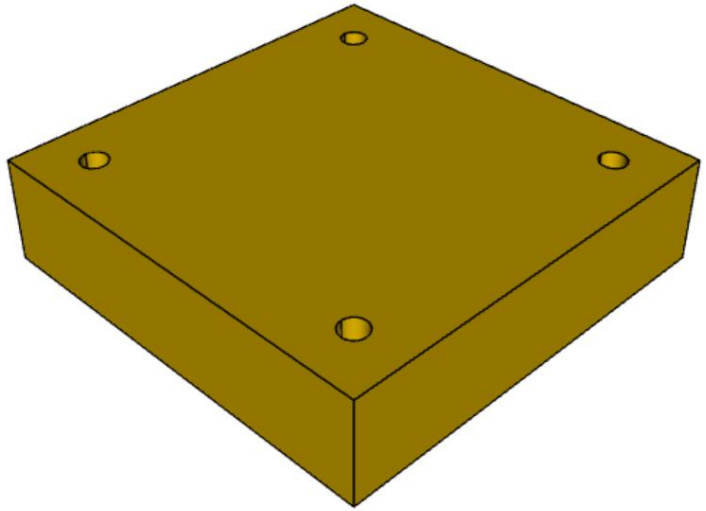


Workplane.transformed()

All Rights Reserved

18 使用构造几何（辅助线）

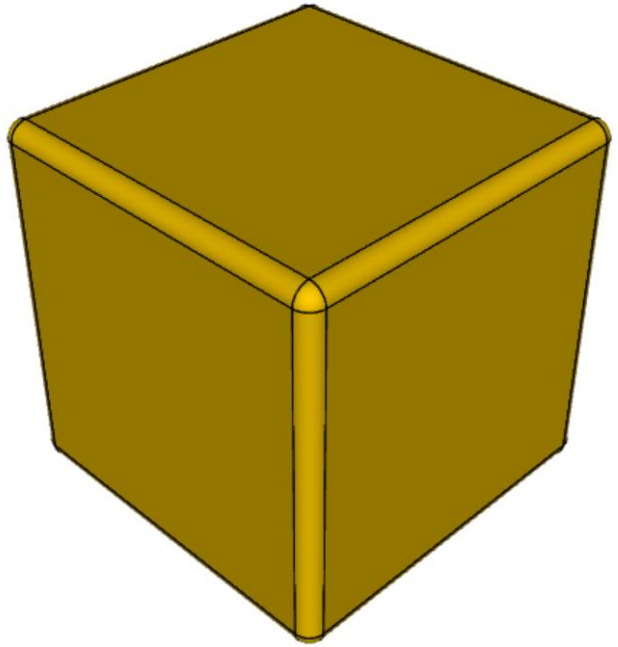
```
result = (
    cq.Workplane("front")
    .box(2, 2, 0.5)
    .faces(">Z")
    .workplane()
    .rect(1.5, 1.5, forConstruction=True)
    .vertices()
    .hole(0.125)
)
```



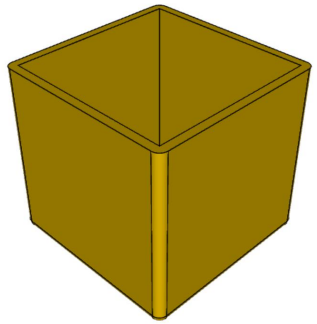
All Rights Reserved

19 抽壳

```
result = cq.Workplane("front").box(2, 2, 2).shell(0.1)
```



Workplane.shell()

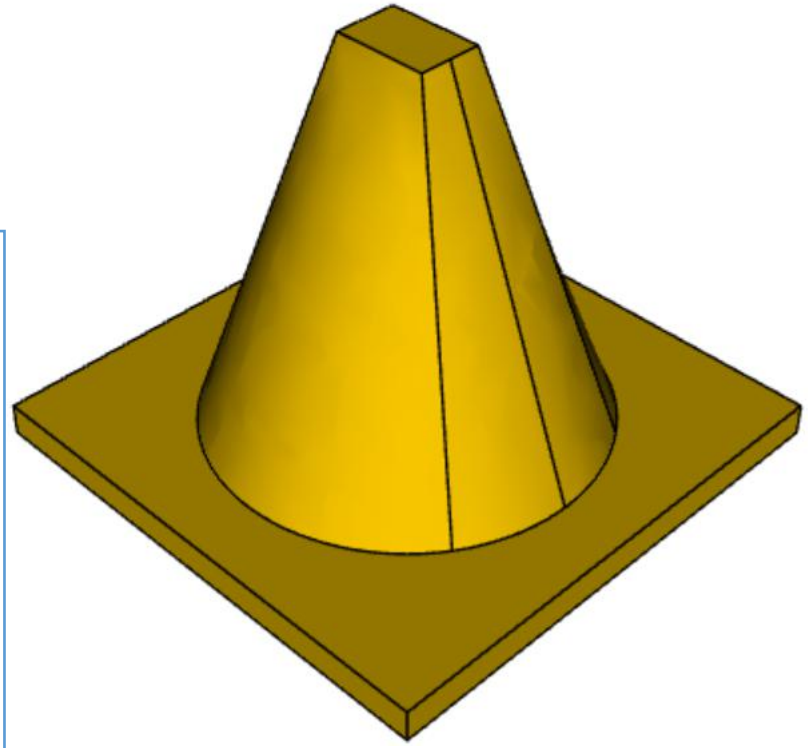


All Rights Reserved

20 放样

此示例在矩形和圆形截面之间创建放样截面

```
result = (
    cq.Workplane("front")
    .box(4.0, 4.0, 0.25)
    .faces(">Z")
    .circle(1.5)
    .workplane(offset=3.0)
    .rect(0.75, 0.5)
    .loft(combine=True)
)
```



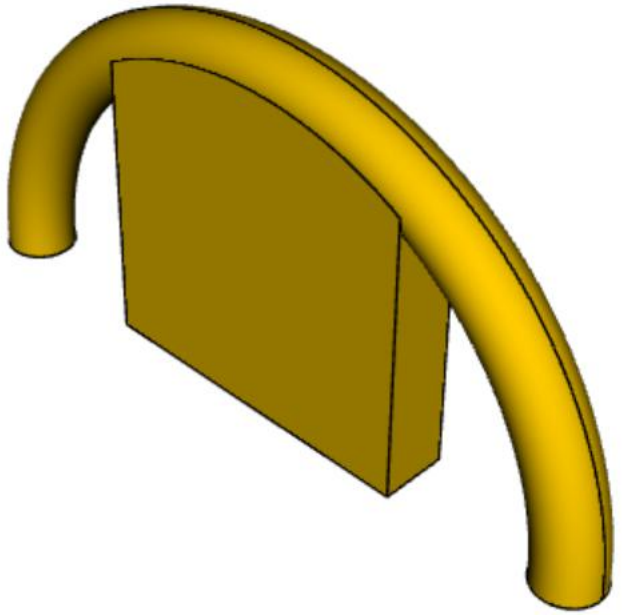
Workplane.loft()

All Rights Reserved

21 挤压到面

有时您会想要挤出一根线，直到给定的面可能不是平面，或者您可能不容易知道必须挤出的距离。

```
result = (
    cq.Workplane(origin=(20, 0, 0))
    .circle(2)
    .revolve(180, (-20, 0, 0), (-20, -1, 0))
    .center(-20, 0)
    .workplane()
    .rect(20, 4)
    .extrude("next")
)
```



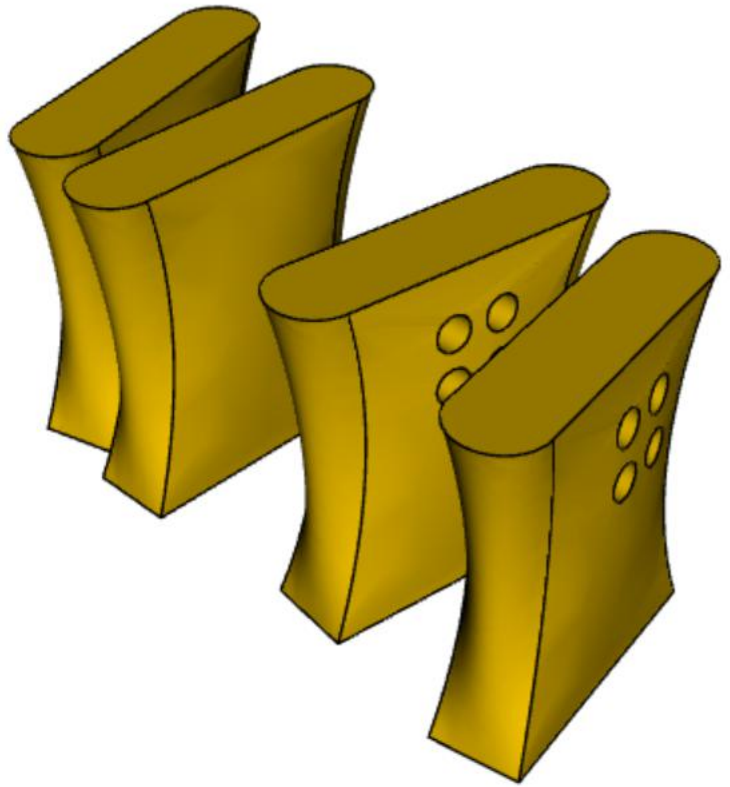
“next”

如果你想要挤出的线不能完全投影到目标表面上，结果将是不可预测的。此外，负责查找候选面的算法通过计算与从线中心沿挤出方向创建的线相交的所有面来进行搜索。因此，请确保您的电线可以投射到目标面上，以避免出现意外行为。

All Rights Reserved

22 延申到指定面

```
skyscrapers_locations = [(-16, 1), (-8, 0), (7, 0.2), (17, -1.2)]
angles = iter([15, 0, -8, 10])
skyscrapers = (
    cq.Workplane()
    .pushPoints(skyscrapers_locations)
    .eachpoint(
        lambda loc: (
            cq.Workplane()
            .rect(5, 16)
            .workplane(offset=10)
            .ellipse(3, 8)
            .workplane(offset=10)
            .slot2D(20, 5, 90)
            .loft()
            .rotateAboutCenter((0, 0, 1), next(angles))
            .val()
            .located(loc)
        )
    )
)

result = (
    skyscrapers.transformed((0, -90, 0))
    .moveTo(15, 0)
    .rect(3, 3, forConstruction=True)
    .vertices()
    .circle(1)
    .cutBlind("last")
)
```

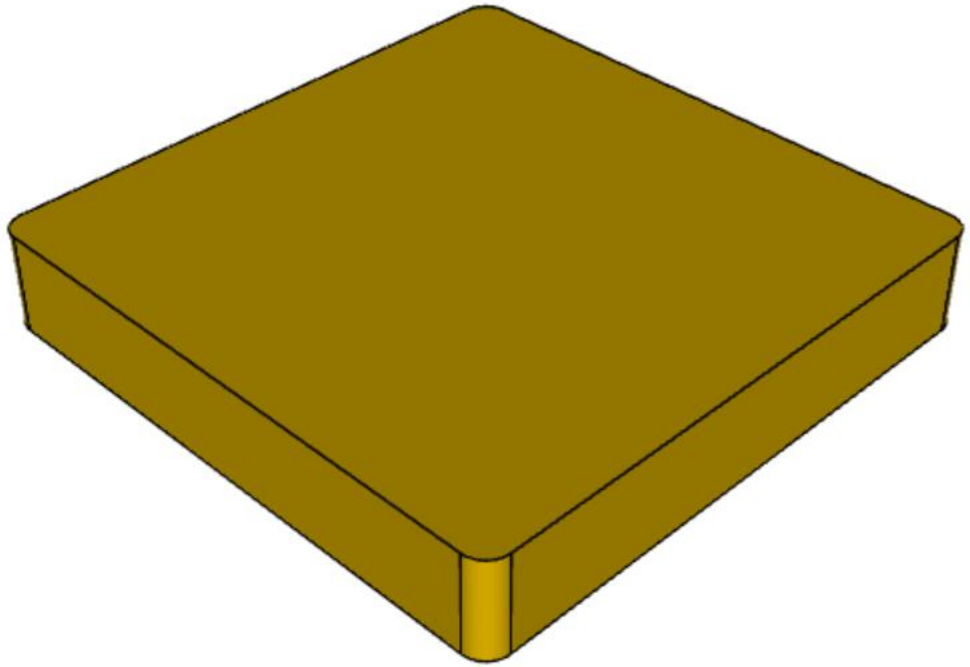


“last”

All Rights Reserved

23 倒角

```
result = cq.Workplane("XY").box(3, 3,
0.5).edges("|Z").fillet(0.125)
```

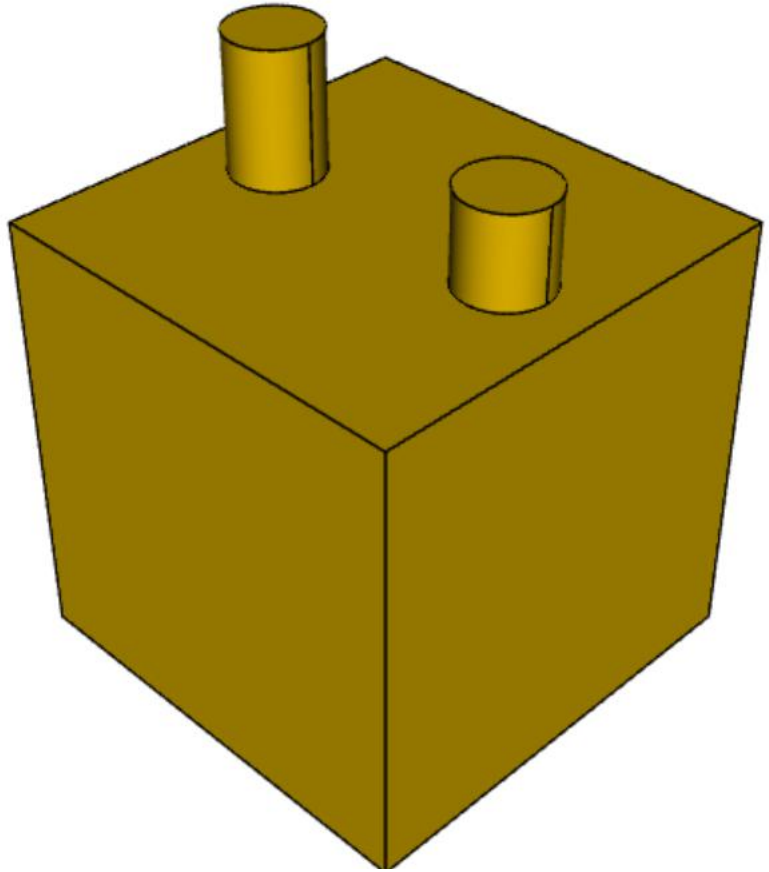


Workplane.fillet()
Workplane.edges()

All Rights Reserved

24 使用标记

```
result = (
    cq.Workplane("XY")
    # create and tag the base workplane
    .box(10, 10, 10)
    .faces(">Z")
    .workplane()
    .tag("baseplane")
    # extrude a cylinder
    .center(-3, 0)
    .circle(1)
    .extrude(3)
    # to reselect the base workplane, simply
    .workplaneFromTagged("baseplane")
    # extrude a second cylinder
    .center(3, 0)
    .circle(1)
    .extrude(2)
)
```

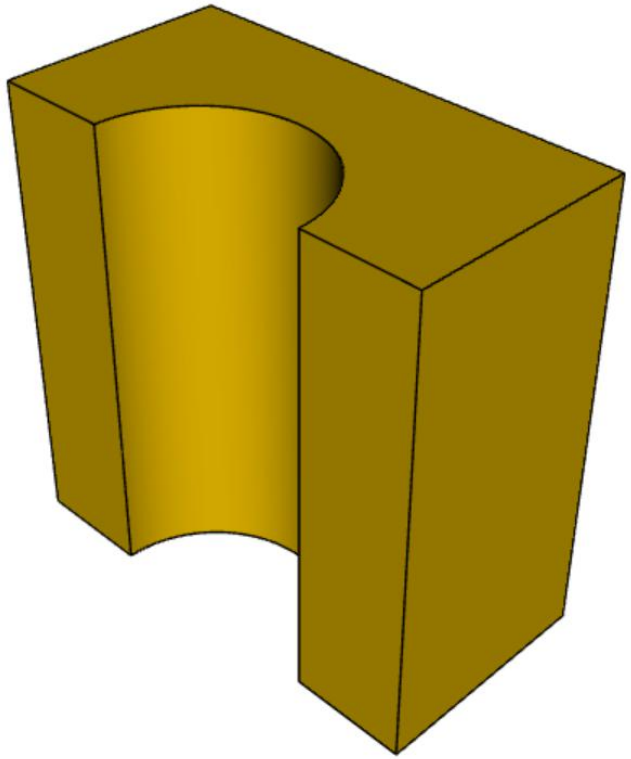


Workplane.tag()
Workplane.getTagged()
Workplane.workplaneFromTagged()

All Rights Reserved

25 切割对象

```
c = cq.Workplane("XY").box(1, 1,
1).faces(">Z").workplane().circle(0.25).cutThruAll()

# now cut it in half sideways
result = c.faces(">Y").workplane(-0.5).split(keepTop=True)
```



All Rights Reserved

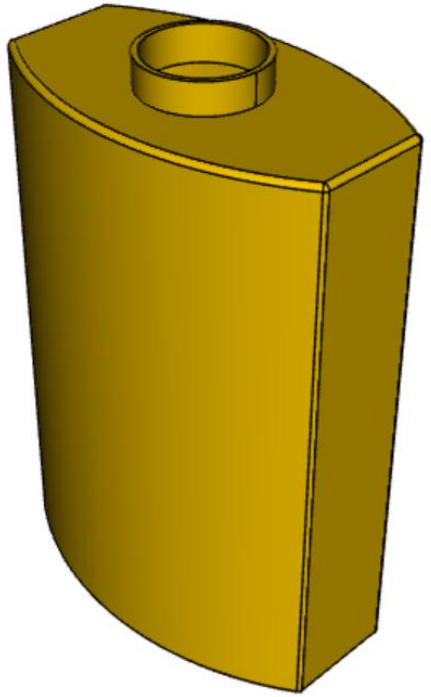
26 构建复杂的例子(摆线齿轮)

```
from math import sin, cos, pi, floor

def hypocycloid(t, r1, r2):
    return (
        (r1 - r2) * cos(t) + r2 * cos(r1 / r2 * t - t),
        (r1 - r2) * sin(t) + r2 * sin(-(r1 / r2 * t - t)),
    )

def epicycloid(t, r1, r2):
    return (
        (r1 + r2) * cos(t) - r2 * cos(r1 / r2 * t + t),
        (r1 + r2) * sin(t) - r2 * sin(r1 / r2 * t + t),
    )

def gear(t, r1=4, r2=1):
    if (-1) ** (1 + floor(t / 2 / pi * (r1 / r2))) < 0:
        return epicycloid(t, r1, r2)
    else:
        return hypocycloid(t, r1, r2)


result = (
    cq.Workplane("XY")
    .parametricCurve(lambda t: gear(t * 2 * pi, 6, 1))
    .twistExtrude(15, 90)
    .faces(">Z")
    .workplane()
    .circle(2)
    .cutThruAll()
)
```

All Rights Reserved

27构建复杂的例子（OCC瓶子）

```
(L, w, t) = (20.0, 6.0, 3.0)
s = cq.Workplane("XY")

p = (
    s.center(-L / 2.0, 0)
    .vLine(w / 2.0)
    .threePointArc((L / 2.0, w / 2.0 + t), (L, w / 2.0))
    .vLine(-w / 2.0)
    .mirrorX()
    .extrude(30.0, True)
)
p =
p.faces(">Z").workplane(centerOption="CenterOfMass").circle(3.0).extrude
(2.0, True)
result = p.faces(">Z").shell(0.3)
```

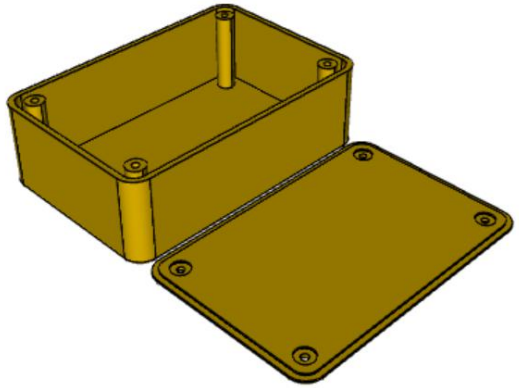


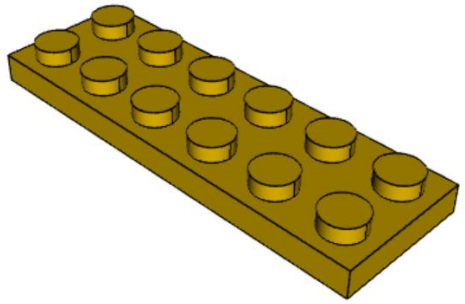
至少比直接使用OCC精简十倍

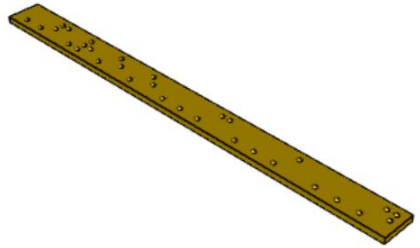
All Rights Reserved

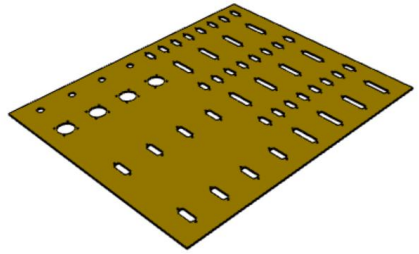
28 更多例子

直接构建复杂模型通常比较困难往往需要复杂的模板组合或者约束求解构建装配体。
更多范例参见：
https://cadquery.readthedocs.io/en/latest/









All Rights Reserved

谢谢！