数学建模 D 赛题，团队控制号 IMMC22099456

一．摘要（D 赛题）

近来，元宇宙概念受到了持续广泛的关注，针对元宇宙的定义各有说辞，但是业界公认的共同点之一就是能够在虚拟世界中创造虚拟物品。理论上讲，用户能够在元宇宙里创建任何物品，但是在实现过程中，设计者需要考虑硬件条件，尤其是计算机存储空间与计算机芯片计算速度的限制。故此我们需要合理设计一组基础构件，用户既能用它们创造想要的物品，又能够最大限度节省计算机存储空间和计算资源。

我们首先将研究对象限定在数量最多的人群、建筑、交通工具和地形。然后再把结果推广到整个香港。为了找到能够组建这些物品的最小构件，我们率先考虑不同房型之间的最大共同构件，以此构件为基础，既能组成包含所有大小的建筑；又作为最大的共同部件，能够尽可能减少使用数量，从而减少计算机内存与计算资源。寻找共同构件的过程，我们通过观察地图，将三维实际物体首先二维平面化，得出建筑只需要三棱柱和长方形两种形状，区别只在于高度不同。

然而由于计算机存储的特性，搭建新建筑时引用构件和创建构件都需要占用内存，当引用太多构件时，不如直接创建一个新的基础构件可以更加节省存储空间。文中通过计算可得，当一个新建筑需要用大于或者等于 49 个基础构件组成，那不如选择将这个新物件建立成为一个新的基础构件，由此节省内存。下文都将这个方法简称为“49 模型”。因此对于特异性较强的人和交通工具，都将创造一个独特的基础构件。

为了提升“49 模型”的全面性，从对单个物体的分析上升到与其他物体共同分析，使其更有普适性，我们又创建了冗余模型。它可以准确地判断需要何时将两个构件合并成一个新的基础组合构件，即当所有组合体中构件和减去组合体的数量差大于等于 48 时，不如新建一个基础构件更加节省空间。

为了使我们构建的虚拟香港更加符合真实情况，我们还将做出虚拟香港的时间演化模型。即每隔一定时间对香港情况重新建立一组数据库。由于计算机内存限制和实际应用场景，我们选择一分钟为时间间隔，模拟每一分钟更新虚拟香港需要的数据存储。

存储计算中我们分成了建立城市所需资源和更新城市所需资源。实际情况中地形和建筑属于静止物品，不需要额外更新数据，变化的只有交通工具和行人。最终计算出建造城市共需 53.78MB，每分钟更新数据运算 562,316,299 次，要使用 136.575MB 空间用于计算，保存数据需要 60.2MB，每天用于数据保存共需 84.7GB。

由此已经建立一个实时的虚拟香港，在如今热门的孪生数字城市[1]中具有广阔的应用，不但可以方便城市规划，还可以为无人驾驶以及短期城市发展预演提供技术基础。

目录

二．建立基础构件

2.1 核心模型

2.1.1 基本的“49 模型”

1．作用：得出需要设置新构件和继续引用已有构件的临界值。

2．思路：我们为了创建出最少数量的构件，所以想要找到需要创建新构件的临界值，由此建立了“49 模型”。

3．定义的参数：

参数	备注
USE_NEED_BYTE 找到并引用一个构件所需的字节	在 2.1 中记作 X
CREATE_NEED_BYTE 创建一个新构件所需的字节	在 2.1 中记作 Y
USE_NUMBER 引用的构件数量	在 2.1 中记作 N
字节 Byte	在全文中记作 B
比特 bit	在全文中记作 b
ASM_NUMBER 组合体的数量	在 2.1.4 中记作 K
SUM_NUMBER_ASM 一个组合体中所有构件的数量和	在 2.1.4 中记作 M
堆叠的层数	在 2.3.4 中记作α
每个三棱锥的底面积	在 2.3.4 中记作 S
每个三棱锥的高	在 2.3.4 中记作 h
每个三棱锥的体积	在 2.3.4 中记作 V
基础三棱锥的个数	在 2.3.4 中记作β
合并的大三棱锥的底面积	在 2.3.4 中记作δ
合并的大三棱锥的高	在 2.3.4 中记作γ
山的高度	在 3.2.3 中记作 k

4．分析：当创建一个新构件所需字节小于继续引用原有构件的字节时，创建新构件更省空间；当创建一个新构件所需字节大于继续引用原有构件的字节时，则继续使用原有构件更省空间。

5．解出临界值的过程：

（1）列出不等式（N 为何取值可创建新构件）：列出的不等式为：NX>Y+X（Y 为定义参数所需字节，X 为引用这个 Y 所用字节）。

（2）X=坐标$5\frac{3}{8}$B（X：0~131,072；Y：0~65,536；Z：0~1,024；精度为 1m）[3]+构件索引号$\frac{7}{8}$B+旋转角度 2B（精度为 2；180 转换为二进制占 1B，两个轴都有旋转参数，所以共 2B.）=$8\frac{1}{4}$ B.

（3）Y=底 388 B（实践得：100×100px 的 *png* 图像占用 388 B）+高$1\frac{1}{4}$ B（香港

最高点“大帽山”为 957m，向上找最接近的 2 的幂，得 1024 m，占 10 b=$1\frac{1}{4}$ B）+高的精度标志符$\frac{1}{8}$ B（1b,为 0 时精度为1m，为 1 时精度为 0.1m）=$389\frac{3}{8}$ B

（4）求出临界值 N：

1. 当 N·X>X+Y 时，创建一个新构件所用的存储空间比用已有构件搭起来所用的存储空间少。

2. 当 N·X<X+Y 时，创建一个新构件所用的存储空间比用已有构件搭起来所用的存储空间多。

3．当 N·X=X+Y 时，可以解得临界值。

解得，$N=48.181818\cdots$。

即，临界约为 48.18。

2.1.2“49 模型”的解释

1．X 的值为何不变：引用任何已创造的元素所用的字节数都相同，都只包含位置、索引、旋转参数这 3 个基本要素，所以它们的字节数相同。

2．Y 的值为何不变：我们定义一个柱形元素，需要知道底和高，这些在 2.1.1.4.3 中已经说过，所以存储空间一定。

3．当一个物体（一般情况下为柱体，但极特殊的情况下可以不为柱体）需要用 N 及以上个构件搭建时，可以另设计一种构件（此处可结合 2.1.4 的想法，让单独的物体与其他相结合，得出最好的设置构件方法）。

2.1.3“49 模型”的优势

“49 模型”可以准确的找出是否需要增加新的构件的临界值，准确方便地辅助人们建立基础构件，使设计出的基础构件科学合理。

2.1.4“49 模型”有待改进的地方

若两个物体的 N 都大于等于 49，而增加两者共同需要的构件可能比分开设立新构件省空间。

2.1.5 拓展的冗余模型

1.简介：基于“49 模型”，我们将模型扩大了范围（不但只对一栋建筑考虑，可以针 对整个香港），创建了改进版，即冗余模型。

2．作用：计算出合并构件和继续用原有构件的临界值。

3. 分析：当合并一个构件所需字节小于继续引用原有构件的字节时，合并新构件更省空间；当合并一个构件所需字节大于继续引用原有构件的字节时，则继续使用原有构件 更省空间。

4. 解出临界值的过程：

（1）列出不等式（N 为何取值可合并构件）：列出的不等式为：KMX>Y+KX。

（2）带入 X 与 Y（由 2.1.1 可得 X=$8\frac{1}{4}$B，Y=$389\frac{3}{8}$B）。

（3）化简不等式，得：KM-K>47.2。

（4）求出 K 与 M 的临界关系：

1.当 KM-K>47.2 时，合并一个构件所用的存储空间比用已有构件搭起来所用的存储空间少。

2.当 KM-K<47.2 时，合并一个新构件所用的存储空间比用已有构件搭起来所用的存储空间多。

3.当继续引用已有构件所用存储空间大于合并一个构件所用存储空间时，最好合并构件。

5．此模型对“49 模型”的补充：

此模型可以用在全局，让“49 模型”从对单独构件进行分析变为与其他构件共同分析。增添了模型的准确性，使单独个体能与其他构件相联系，从而具有适用性。

2.2 核心模型建立基础构件的实际应用

1．由于 48.18 是临界值 N，则如果要构建新的构件，那以 49 已有构件为单

位的构件是最好的。因为若构建一个小于 49 的构件，则由核心公式可得，不如用更小的构件搭建；若构建大于 49，则构建 49 的构件便只能用 49 块及以上更小的构件搭建，则没有建立一个新构件节省空间。（此结论会在 2.3 中频繁使用）。

2. 在经过 2.3 后，我们会得出许多基本构件。此时若再出现复杂的柱体，我们可以先将其看做一个平面图形（直柱体同一平面上图形相同），再基于“49 模型”用已有图形拼凑若需要 49 块及以上的图形，则可以新建一个构件。这正是一个将三维问题转化为二维问题，再将二维的结论转化为三维结论的思路。（此处因为时间问题和考虑到一般计算机的储存空间，我们在后文没有考虑复杂的柱体的拼凑）。

2.3 对香港元素进行分类

通过观察谷歌地球卫星图[3]，我们发现香港最主要的元素是地形（主要是山、水）、建筑物、交通工具、人，这也是我们建立数字孪生城市最主要的研究对象。因此我们对这四种元素进行了建构，得出了以下的 88 个基础构件。

2.3.1 交通工具

交通工具是组成城市的重要一部分，追踪行驶路线也是我们建立孪生城市的目的之一。因此我们需要在虚拟城市中加入交通工具。在这里我们只考虑了地上可见的交通工具，未考虑地铁、飞机等其他交通工具。由于车辆的构造非常复杂，需要的基础构件已经远大于 49 个，故根据“49 模型”需要创建一个新的构件（有关内容参见 2.2 核心模型）。城市里常见的车辆主要分为私家车、大型客车、警车、救护车、消防车、工程车和自行车，共 7 种构件。

综上所述，交通工具我们共创建了 7 种构件。

2.3.2 建筑

建筑也是城市中重要的组成部分。现实中的构成建筑的图形有多种形状，如圆柱、正方体、三棱柱、不规则图形.....为了减少储存空间而不影响数字孪生城市的作用，我们将虚拟香港中所有建筑都近似看成柱体再加上一些细节，包括

正方体、长方体、正三棱柱。此处我们并未考虑圆柱，因为不同尺寸的圆柱体无法和以上三种柱体一样无缝拼接。若考虑圆柱体，则需要创建无数个不同尺寸的圆柱体构件。建筑中的曲面都可近似为其他立体图形或它们的组合体。

因为我们在2.3中建立的基础构件大多为直柱体,所以在面对三维的图形时，我们可以仅仅将注意力放在底面上，确定几个基本地平面图形。从高为一米（我们设立的精度为一米）开始，直至香港最高楼环球贸易广场 484 米[4],通过调节高的参数且遵循“49 模型”，每用到 49 个基本构件就重新创建一个或一组新的基础构件。以此来创建不同高度但底相同的基础构件。又根据[code:loop-best-height3.py]的计算，我们确定了 1m，5m 和 22m 是最节省空间的基础构件高度。

因为香港最宽建筑香港国际机场航站楼宽度约为 1000m[6]，不足 2401（7^4）m，所以长方体最大底面积为 49m*49m。因此，我们可以得到如下图九种基础长方体构件（未考虑颜色）。

尺寸（前两个乘数为底边长，最后一个乘数为高）	建筑样式	建筑颜色[5]
1m*1m*1m	一般建筑	红/橙/蓝/银白
1m*1m*5m		
1m*1m*22m		
7m*7m1m		
7m*7m*5m		
7m*7m*22m		
49m*49m*1m		
49m*49m*5m		
49m*49m*22m		

如上图，基础长方体构件（考虑颜色）共有 9*4=36 个。

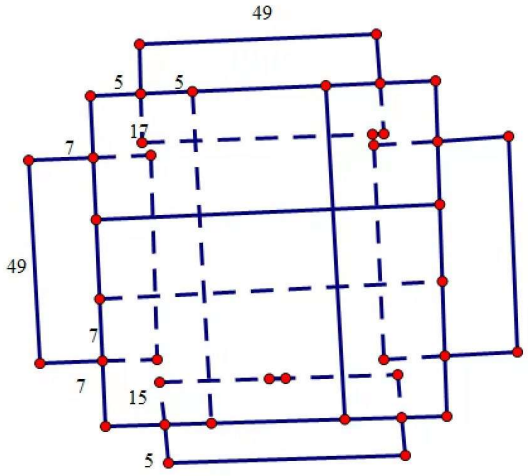
同理，基础三棱柱构件（1m*1m*24m 即底面边长为 1m，高为 24m）也有 36 个。

综上所述，用于构建建筑的基础柱体构件共 36+36=72 个。

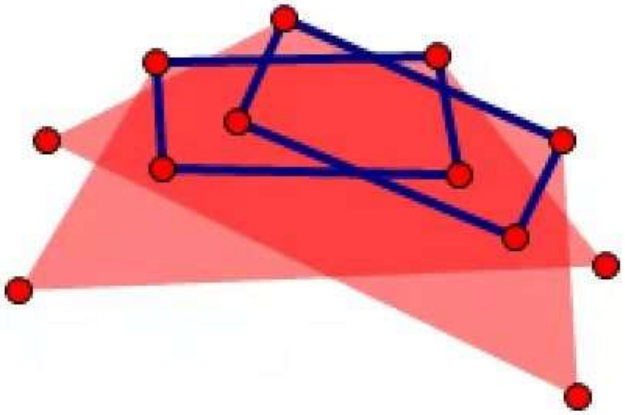
为了验证我们模型所建造的构件的实际可信性，我们精心挑选了一栋建筑——香港如心广场。这栋建筑既有高楼，又有较矮的底座，还有圆柱和圆环，几乎模拟了所有的建筑类型。

我们把它拆成了 3 个部分——高座、低座、底盘，分别进行建模。

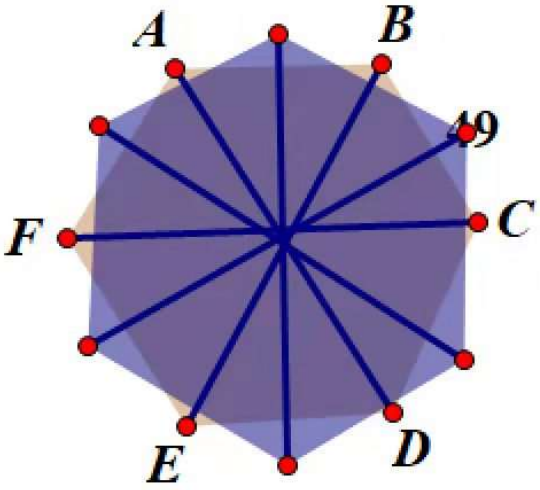
（1）高座：高座几乎是一个长方体，但是我们还是顾及到了边上突出的部分。在平面上的布局是类似这样的：



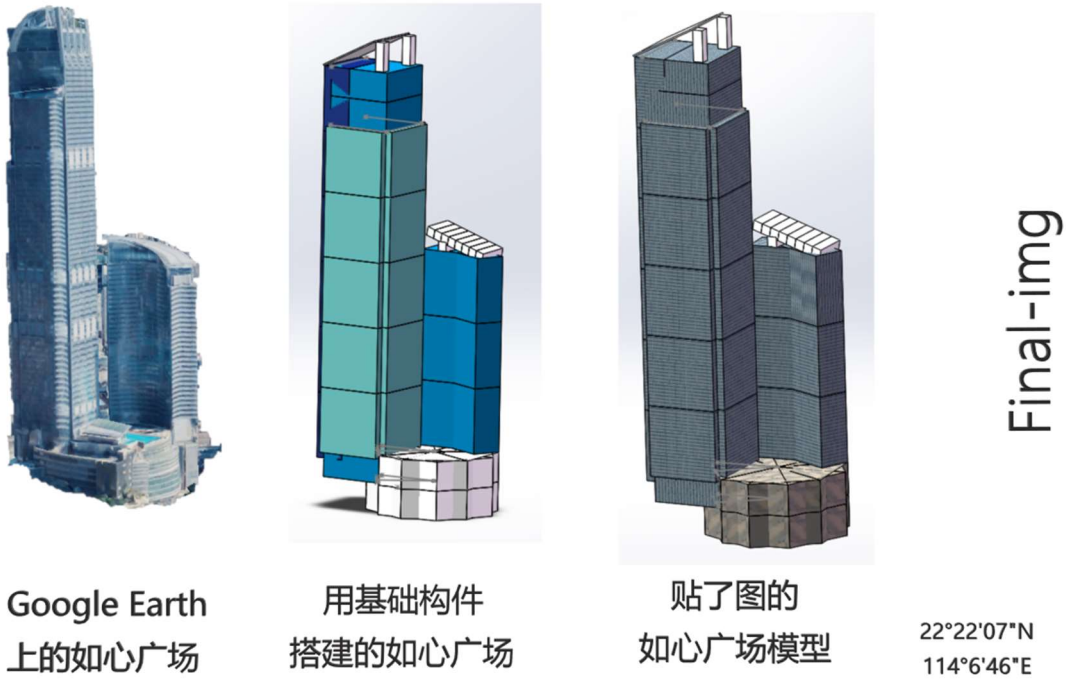
（2）低座：低座的底面可以看成是一个 1/4 圆环。我们的构件库中没有圆环，所以我使用了两个长方体相交叠，如图：



（3）底盘:底盘可以看成近似的圆柱体，但是在构件库中没有圆柱体，所以我们采取了“近似”的方法。我们用 6 个正三棱柱拼成了一个六棱柱，然后两个六棱柱相重合，稍微错开，如图：



完成后，我们又在上面加了一些小零部件，再组合起来。下图是我们通过 Solid Works 软件用我们所设计的基础构件所搭建出的香港如心广场[3]模型：



2.3.3 图

2.3.3 地形

我们考虑的因素还有地形，地形也是城市中重要的元素，城市的规划很大程度上都受到地形的影响。地形中最主要的是水域和陆地。陆地又可分为平原和山地，由于在寸土寸金的香港，大部分平原都已被建筑物覆盖，因此我们只考虑了山地。所有山都大致可以近似看作一个正三棱锥。正三棱锥的好处就是可以用若干小三棱锥堆叠成大三棱锥，所以我们可以建立一个方程求出堆叠所需的小三棱锥数量：

设堆叠的层数为α，每个三棱锥底面面积为 S，高为 h，体积为 V，总共会用β个基础三棱锥，合并的大三棱锥底面面积为δ，高为γ

可得

$$\delta = S\left[1+3+\cdots\cdots+(2\alpha-3)+(2\alpha-1)\right]=\alpha^2 S$$

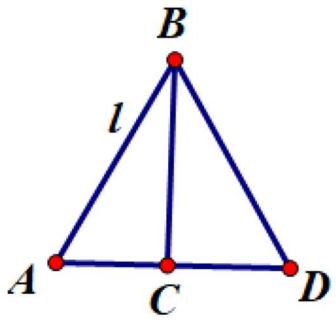
$$\gamma=\alpha h$$

$$V = Sh$$

$$\beta=\delta\gamma/V=\alpha h\cdot\alpha^2\cdot\frac{s}{sh}=\alpha^3$$

通过我们的核心模型（2.1），可以得知，当 β>48 时，创造一个新基础构件更节省储存空间。

因为 $\sqrt[3]{48.2}$ ≈3.64，所以当大三棱锥的棱长约为小三棱锥的 3.6 倍时最节省储存空间，但由于 α ∈N^+，α=3 或 4 倍时即为最佳，又因为香港最高点大帽山山顶海拔约为 957m[2]，且正三棱锥高为棱长的 $\sqrt{\frac{3}{4}}$ 倍约为 0.866 倍，下面给出证明：



设正三棱锥棱长为 L，
则此时高可以近似的看作在一个边长为 L 等边三角形的高，

AB = L，AC = $\frac{L}{2}$，

∠A = 60°，

AB : BC = sinA = 2 : $\sqrt{3}$

BC = $\frac{L}{2}\times\sqrt{3}$ = $\sqrt{\frac{3}{4}}$L 约等于 0.866L

根据[Code:best-height.py]及输入的数据(数据见附表)可得，取棱长为 1m，4m，12m，36m，108m，324m，972m 为最佳，又因为山体有 1 种常见颜色，即绿色，所以山地我们一共创建了 7×1=7 种基础构件。

水域主要可分为河流、湖泊、海洋，由于海洋不在我们虚拟城市的研究范围内，所以我们暂不考虑。香港的湖泊面积都较小，因此我们可以用蓝色的长方体（和搭建建筑的构件相同，所以不需要设计新的构件）构件来搭建湖泊。对于河流，我们同样可以用蓝色的柱体构件搭建出来（香港最长河流深圳河[7]大约需要 44 个基础长方体构件，并未达到 49 个，不需要创建新的构件），所以水域我们不需要重新创建构件。

综上所述，地形部分我们共需要创建 7 种基础构件。

2.3.4 人

人也是城市中不可或缺的。和 2.4.1 交通工具相似，人同样难以用少于 49 个的基础构件构造出来，因此我们需要创建一类新的基础构件——人。关于人的分类标准，我们有很多想法（性别，人种，国籍……），最终确定为市民/非市民。因为按照这样的分类标准，我们可以更好地管理人民（市民和非市民的税务等多方面都有所不同），以达到我们创建虚拟城市的目的。为了直观的区分市民和非市民，我们让市民的构件胸前别上香港特别行政区的区徽，同时也可以用颜色区分。所以此处我们一共建造了 2 个基础构件。

2.4 总结

在本节中，我们的任务是通过尽量少的基础构件在虚拟世界（即元宇宙）中搭建一个虚拟香港。通过查询相关资料[1]，我们可以得知，这是为了搭建数字孪生城市而设计的。数字孪生城市主要是为了赋能真实物质城市，在实时预测、管理规划等方面为人们提供平台。为了确定基础构件，我们需要找出需要增添新构件的临界值，于是便创建出了“49 模型”，即若一个物体是由大于等于 49 个基础构件搭建成的，则需要为它创建一个新的基础构件。接着，为完善“49 模型”，我们进一步推论出了冗余模型，即若所有组合体中构件和减去组合体的数量差大于等于 48 时，则需要将两个基础构件合并。通过上述模型，我们可以得到，虚拟城市中最主要的元素是地形（即山地，水域）、建筑物、人、交通工具。我们分别对其进行了模拟，通过观察与 Solid Works 的实际搭建，我们设计出了不同种类、颜色与大小共 88 个构件。为了验证我们的模型，我们通过 Solid Works 软件，使用我们的基础构件，完美地搭建出了在香港精心挑选的一栋复杂建筑——香港如心广场。这在很大程度上证明了我们的模型所设计的基础构件是切实可行的。不仅限于题目，我们还考虑了我们的模型在现实中的意义。它可以准确便捷地找出事物需要发生变化的临界值，从而从本质上帮助人们简化问题。如果时间充裕，我们还可以将模型应用到更多物品中，将人物模型做得更加精致，增加更多动物模型。

三．保持与真实香港实时同步的虚拟香港

为了更好地达成我们建立数字孪生香港的目的，我们需要让虚拟香港以尽可能短的时间间隔与真实香港实时同步。同步所需的资源是非常多的，在本节中，我们将会计算出建立虚拟香港以及将虚拟香港与现实香港同步所需的计算资源和存储资源，并在切实可行的基础上，确定一个适合的同步时间间隔。

3.1 确定时间间隔

我们建造的模型是为了帮助城市管理者监控城市、疏导交通、流量分析甚至让用户“云游览”。

所以更新时间不能过长，否则就达不到交通疏导的"实时"、监控的"连续" 等要求；也不能过短，否则计算机会不停地计算，但很多数据间隔过小，没有太大的意义，造成资源的浪费。

综上所述，我们认为每分钟更新一次是比较好的。

这样既可以做到连续的记录（1 分钟，汽车不会跑太远，况且我们还有车牌号），同时也可以达到实时监控的时间要求。

3.2 建造和更新城市所需存储资源

3.2.1 存储资源的分析

建立城市所需的存储资源，可以分为两部分，即建立所有基础构件所需资源，以及在建立城市时引用构件所需资源。

由前文（2.1.1 基本的“49 模型”）可知，建立单个基础构件所需的计算资源为 389.25B，根据 2.4 可知，建立虚拟城市所需的基础构件共有 88 个。因此，建立所有基础构件所需资源为 389.25∗88 = 34,254 B 即 $\frac{34254B}{1024^2} \approx 0.03MB$ 。

建立城市时引用的构件，包括建筑物构件（柱体）、地形构件（锥体）、人物构件、交通工具构件。

3.2.2 对于建筑构件的引用

我们在香港随机选取了 13 栋建筑，将它们近似看作长方体，并估算出它们的长、宽、高。我们将这些建筑分为三类，即“大”（长、宽、高中有任一条边长>=100m 或长、宽、高均>20m）、“中”（未被归为“大”、“小”的建筑）、“小”（长、宽、高均<20m）。通过查询资料，我们得知香港的“大”、“中”、“小”三类建筑之比大致为 4:5:1。因此，我们为“大”、“中”、“小”三类建筑分别赋予权重，即“大”类权重为 0.4，“中”类为 0.5，“小”类为 0.1。

我们通过计算机模拟搭建，得出每栋“大”类建筑所需基础构件的平均值为 26.4（又因为我们考虑百米及以上建筑时只能将它拆开拼装，而且 100~200m 的建筑相对较多，400m 及以上建筑相对较少，此时我们取平均系数 2.5，所以“大”

类建筑所需基础构件平均值为 66），“中”类建筑为 19.5，“小”类建筑为 26。因此，我们可以得出，所有建筑所需基础构件平均约为

66×0.4 + 19.5×0.5 + 26×0.1 = 38.75 个。

由于香港共有约 42,000 栋建筑[8]，所以我们总共需要的基础构件数为 38.75×42,000=1,627,500 个。又因为找到并引用一个基础构件所需要的存储资源为 8.25B（见 2.2.2 核心模型的解释），所以搭建所有建筑所需存储资源为 $1627500*8.25B=13426875B$ 即 $\frac{1342687B}{1024^2}\approx 12.8MB$ 。

（具体表格见附录。）

3.2.3 对于地形构件的引用

以下是我们所选出的三棱锥棱长和高的对照表。

编号	a_1	a_2	a_3	a_4	a_5	a_6	a_7
棱长	1	4	12	36	108	324	972
高	0.9	3.5	10.4	31.2	93.5	280.6	814.8

等边三角形的高与边长的比为 $2:\sqrt[2]{3}$，所以高约为棱长的 0.866 倍。

我们进行如下操作，设山的高度为 k 米且 $a_i < k < a_{i+1}$（i 为不大于 6 的正整数）。

（1） $a_i < k < 2a_i$

则用 8 个棱长为 a_i 的拼成棱长为 $2a_i$ 的三棱锥，可以进行如下操作，将各个面均朝着中心平移，可将对应面所有对应棱棱长缩小，即可得出棱长为 k 的三棱锥。

（2） $2a_i < k < 3a_i$

用 27 个棱长为 a_i 的拼成棱长为 $3a_i$ 的三棱锥，进行（1）中的操作即可得出棱长为 k 的三棱锥

并且 $50 < k < 957$，故而此结论成立

为了方便统计，我们采取了抽样调查的方法。我们随机选取了香港的 5 座山，经过简单的求平均数计算不难算出，平均每座山所需要的方块数约为 19.4 块。下面为抽样出的五座山的所需方块与海拔高度表。

山的名称	海拔高度	构件件数
狮子山	495m	8 个
凤凰山	934m	8 个
马鞍山	702m	27 个
草山	647m	27 个
吊手岩	588m	27 个

香港的山有 310 座[9]，可以算出共需 310*19.4 = 6014 块基础构件，而应用每块基础构件需要 8.25B。所以总字节数为 6014*8.25B = 49,615.5B ≈ 48.5KB ≈ 0.05MB。

3.2.4 对于人物构件的引用

我们将每一个表示人的数据串分为坐标和是否是市民的判断。坐标共需要占用约 5.375B 的存储资源（计算过程详见 2.1.1）。一个人是否是香港市民，我们可以用二进制中的 0 和 1 来判断，此为 1 bit，即 0.125 B 的存储资源。所以一个人所需的存储资源大约为 5.375 B+0.125 B = 5.5 B。通过查询资料，我们得出了香港的人口大约为 739.47 万人[11]。因此，所有香港人口所需存储资源为 7,394,700 * 5.5 B = 40,670,850 B 即 $\frac{40670850B}{1024^2}\approx 38.8\text{MB}$。

3.2.5 对于交通工具构件的引用

我们将表示交通工具的数据串分为坐标和车牌。坐标共需要占用约 5.375 B 的存储资源。香港车牌可以分为两部分，字母部分和数字部分[12]。字母部分总共有 26 个字母（即可以用 2^5 涵盖），所以每个字母需要 5 b，两个字母则需要 10 b。数字部分总共有四位，从 0000 到 9999，共 10,000 种（即可以用 2^{14} 涵盖）。则需要 14 b。则数字和字母共需 24 b（即 $\frac{24}{3}=8B$ ）。通过查资料得出香港大约有 73.7 万辆车[10]。

则香港所有交通工具的存储资源为

$$737000 * 3 \text{ B} = 2,211,000 \text{ B} \approx 2.1 \text{ MB}$$

3.2.6 建造城市总共需要的存储资源

0.03 MB + 12.8 MB + 0.05 MB + 38.8 MB + 2.1 MB = 53.78 MB

3.2.7：更新城市所需存储资源

一次更新，要存储 60.2 MB 的资源，若每分钟更新一次，则一天更新 24*60 = 1,440 次，需要存储 1,440*60.2 MB = 86,688 MB ≈ 84.7 GB，即每天存 储 84.7 GB 的存储空间。

一个 1TB 的移动硬盘大概要 250 元人民币，平均每 G 约 4 元人民币。

这样算，每天约需要 338.8 元人民币的存储空间费用。

3.3 实时更新城市所需计算资源

3.3.1 计算资源的分类

计算资源分为 CPU 资源和内存资源两类，所以我们需要分别对它们进行计算，最终计算资源会得出两个结果。

3.3.2 更新过程中 CPU 的运算次数

更新步骤：

1. 分割字符串并存储到变量：

有 8,131,700 个数据小段，所以要分割 8,131,699 次。存储到变量，需要经历以下步骤：获取存储地址->存储，共要计算（1+数据长度+1）= 2+数据长度 次操作。每个数据小段约为：

7,394,700 个“人”变量，每个占 5.5+2 =7.5 B；

737,000 个“车”变量，每个占 8.375+2 = 10.375 B.

也就是说进行 8,131,699 + (2+60)*7,394,700 + (2+83)*737,000 = 472,867,599 次运算。

2. 删除上一次的更新：

分为两步，寻找->删除->存储。

上一次更新了 8,131,700 个数据小段，所以要重复 8,131,700 次，

总共运算 8,131,700*3 = 24,395,100 次。

3.解析数据、绘制模型：

提取数据（四段数据，4 次）->解析坐标（3 个坐标轴，3 次）->解析特征(如'是不是市民'，1 个判断，1 次)->绘制模型（1 次），共 4 步；

每一步进行 4+3+1+1=9 次运算。8,131,700 个数据小段，共运算 8,131,700*9=73,185,300 次运算。

综上所述，共要进行 472,867,599 + 24,395,100 + 73,185,300 = 570,447,999 次运算。

3.3.3 所需内存资源

存储一串数据（变量）至少需要：

1.指针，指向数据串起始内存地址（约 1B）

2.数据串

3.结束标志符（约 1B）

总计应为（数据串长度+2）B

更新步骤：

1.分割：划分约 8,131,700 个变量，这些都将存储在内存中，以供 步骤 3 调用。其中约 7,394,700 个为"人"变量，每个占 5.5+2=7.5 B;另约 737,000 个为"车"变量，每个占 8.375+2 = 10.375 B. 总计占约 63,106,625 B ≈ 60.2 MB.

解析：此过程增加内存。

2.删除上一次的更新：需要根据上一次更新所保留的指针，寻找到上次更新的数据；再删除 人/车 的模型，删除索引号，释放内存，但存储数据到硬盘。

解析：此过程释放内存。

3.解析数据、绘制模型：根据 步骤 1 中的变量，提取 x、y、z 坐标（x:17b, y:16b, z:10b），以及“人”的“是否是市民”信息/“车”的“车牌”信息。此时共需要 4 个变量，“人”的共约 13.5 B；“车”的共约 16.375 B。解析完成后，只能释放这一步中的 4 个变量，不能释放模型的内存，因为还要留给 步骤 2 来删除模型。

解析：此过程会适量增加内存，但上限为增加 16.375 B。

总结：因为要同时存储两套更新数据，所以会占用 60.2 MB * 2 = 120.4 MB 的内存。因为解析时会增加使用量，上限为 16.375 B，所以最多同时使用 136.575 MB 的内存。

四. 总结与展望

4.1 总结

在任务二中，我们需要计算虚拟香港与现实香港同步所需的资源，并确定适当的同步时间精度。我们首先确定的是同步的时间精度。同步时间间隔过大会导致数据失去时效性，间隔过小会导致计算机负荷过大，积累较多无效存储。因此，我们选择了一分钟作为同步时间间隔。

同步所需的资源可以分为两类——存储资源和计算资源，其中存储资源可分为建立基础构件所需资源、引用建筑构件所需资源、引用地形构件所需资源、引用人物构件所需资源、引用交通工具构件所需资源；计算资源又可分为内存资源和 CPU 资源。我们对这三类资源分别进行了计算，最终结论为存储资源共需 60.2MB（创建构件需 0.03MB，引用建筑构件需 12.8MB，引用地形构件需 0.05MB，引用人物构件需 38.8MB，引用交通工具构件需 2.1MB），内存资源共需 136.575MB，CPU 资源共需 562,316,299 次运算。

在一分钟的同步时间间隔下，每天所需的内存资源为 84.7GB，折合市价约 338.8 元。

通过计算结果，我们也可以间接验证出一分钟的同步时间间隔是可行的（金费在预算范围内），不过由于时间问题，我们并未在计算机中模拟同步过程。

我们的模型在现实生活中也是非常有指导意义的。任务一中的模型可以帮助人们在现实生活中用尽量少的基础构件，即最节省的计算机存储资源，较完美地搭建出一座虚拟城市。任务二中的模型对任务一已经搭建的模型做出了一个随时间演化的真实模型。可以更加真实地实现真实城市虚拟数字化，仿佛一个孪生城市。由此实现城市道路规划，城市建设设计，城市安全规划以及无人驾驶路线安排等功能。当数字孪生城市日积月累地发展之后，计算机存储资源将会是一个不可忽略的重要因素，因而尽可能地节省单位时间资源是一个具有长远意义的课题。

4.2 展望

在任务一的基础构件模拟中，我们出于时间限制，只考虑了用长方体、三棱锥与三棱柱去模拟建筑、行人与交通工具，但是并没有考虑到更加普遍的圆形结构，导致模拟的实物具有局限性，美观性也不足。同样在针对基础构件的简化中，我们只考虑了两个物体重合可以进行的简化模型，没有考虑多个物体重合的情况，这个可以进一步优化。

任务二的情况我们将建筑与街道当作静止物体，这在短时间内并无问题，但是香港作为高速发展的城市，在以年为时间单位时，建筑和街道也会发生一定的变化。由此可以进一步将模型改进更加符合真实。

五. 参考文献

[1]《数字孪生城市平台原型的初步设想》作者：杨 滔 杨保军 刘 畅 张晔玥
[2]https://baike.baidu.com/item/大帽山/75677 香港最高山
[3]earth.google.com 香港东西、南北跨度，城市四种主要元素，香港如心广场各数据，香港“大”、“中”、“小”建筑之比
[4]https://baike.baidu.com/item/环球贸易广场/10458972?fr=aladdin 香港最高楼
[5]https://baike.baidu.com/item/建筑色彩/2437585 建筑常用色
[6]https://baike.baidu.com/item/香港国际机场/1128141 香港最宽建筑
[7]https://baike.baidu.com/item/深圳河/25344?fr=aladdin 香港最长河
[8] https://www.beamsociety.org.hk/en_index.php 香港建筑数
[9] https://zhidao.baidu.com/question/1238161946553661699.html 香港山峰数
[10] https://auto.china.com/mip/17729.html 香港车辆数
[11] https://user.guancha.cn/main/content?id=574233 香港人口数
[12]https://baike.baidu.com/item/香港车牌/7103762 香港车牌

附录：

附表 1：模型库

编号	模型所属	模型名称	尺寸		颜色	备注
			底面边长	高		
1	四棱柱	1*1*1 正方体	1	1	红	
2					白	
3					蓝	
4					橙	
5		1*1*5 长方体	1	5	红	
6					白	
7					蓝	
8					橙	
9		1*1*22	1	22	红	

10	四棱柱	长方体			白	
11					蓝	
12					橙	
13		7*7*1 长方体	7	1	红	
14					白	
15					蓝	
16					橙	
17		7*7*5 长方体	7	5	红	
18					白	
19					蓝	
20					橙	
21		7*7*22 长方体	7	22	红	
22					白	
23					蓝	
24					橙	
25		49*49*1 长方体	49	1	红	
26					白	
27					蓝	
28					橙	
29		49*49*5 长方体	49	5	红	
30					白	
31					蓝	
32					橙	
33		49*49*22 长方体	49	22	红	
34					白	
35					蓝	
36					橙	
37	三棱柱	1，1 正三棱柱	1	1	红	
38					白	
39					蓝	
40					橙	
41		1，5 正三棱柱	1	5	红	
42					白	
43					蓝	
44					橙	
45		1, 22 正三棱柱	1	22	红	
46					白	
47					蓝	
48					橙	
49		7,1 正三棱柱	7	1	红	
50					白	

51	三棱柱				蓝	
52					橙	
53		7,5 正三棱柱	7	5	红	
54					白	
55					蓝	
56					橙	
57		7,22 正三棱柱	7	22	红	
58					白	
59					蓝	
60					橙	
61		49,1 正三棱柱	49	1	红	
62					白	
63					蓝	
64					橙	
65		49,7 正三棱柱	49	7	红	
66					白	
67					蓝	
68					橙	
69		49,22 正三棱柱	49	22	红	
70					白	
71					蓝	
72					橙	
73	三棱锥	1 正三棱锥	1	0.9	绿	
74		4 正三棱锥	4	3.5	绿	
75		12 正三棱锥	12	10.4	绿	
76		36 正三棱锥	36	31.2	绿	
77		108 正三棱锥	108	93.5	绿	
78		324 正三棱锥	324	280.6	绿	
79		972 正三棱锥	972	814.8	绿	
80	其他	人（市民）	\	\	不同	区分
81		人（非市民）	\	\		
82		私家车	\	\	不同	主要是形状、颜色的不同
82		大客车	\	\		
84		警车	\	\		
85		救护车	\	\		
86		消防车	\	\		
87		工程车	\	\		
88		自行车	\	\		

附表 2：[Code:best-height.py]的输入数据及运算结果

编号	棱长 1	棱长 2	棱长 3	棱长 4	棱长 5	棱长 6	棱长 7	计算结果
	高 1	高 2	高 3	高 4	高 5	高 6	高 7	
1	1	4	16	64	256	768	\	26100
	0.9	3.5	13.9	55.4	221.7	665.1	\	
2	1	4	16	64	192	768	\	25950
	0.9	3.5	13.9	55.4	166.3	665.1	\	
3	1	3	12	48	192	576	\	22254
	0.9	2.6	10.4	41.5	166.3	498.8	\	
4	1	3	12	36	144	576	\	24960
	0.9	2.6	10.4	31.2	124.7	498.8	\	
5	1	3	12	48	144	576	\	24762
	0.9	2.6	10.4	41.5	124.7	498.8	\	
6	1	4	12	48	192	576	\	24372
	0.9	3.5	10.4	41.5	166.3	498.8	\	
7	1	4	12	36	144	576	\	24504
	0.9	3.5	10.4	31.2	124.7	498.8	\	
8	1	4	12	48	144	576	\	24366
	0.9	3.5	10.4	41.5	124.7	498.8	\	
9	1	3	12	48	192	576	\	23742
	0.9	2.6	10.4	41.5	166.3	498.8	\	
10	1	4	12	48	192	768	\	26320
	0.9	3.5	10.4	41.5	166.3	665.1	\	
11	1	4	16	48	192	768	\	25123
	0.9	3.5	13.9	41.5	166.3	665.1	\	
12	1	4	16	64	256	1024	\	22106
	0.9	3.5	13.9	55.4	221.7	886.8	\	
13	1	3	9	27	81	243	729	22106
	0.9	2.6	7.8	23.4	70.1	210.4	631.3	
14	1	3	9	27	81	243	972	22106
	0.9	2.6	7.8	23.4	70.1	210.4	814.8	
15	1	3	9	27	81	324	972	24327
	0.9	2.6	7.8	23.4	70.1	280.6	814.8	
16	1	3	9	27	108	324	972	27321
	0.9	2.6	7.8	23.4	93.5	280.6	814.8	
17	1	3	9	36	108	324	972	21560
	0.9	2.6	7.8	31.2	93.5	280.6	814.8	
18	1	3	12	36	108	324	972	23498
	0.9	2.6	10.4	31.2	93.5	280.6	814.8	
19	1	4	12	36	108	324	972	20930
	0.9	3.5	10.4	31.2	93.5	280.6	814.8	
20	1	3	9	27	108	432	\	21231
	0.9	2.6	7.8	23.4	93.5	374.1	\	

21	1	3	9	36	144	432	\	24320
21	0.9	2.6	7.8	31.2	124.7	374.1	\	24320
22	1	3	9	36	108	432	\	25672
	0.9	2.6	7.8	31.2	93.5	374.1	\	
23	1	3	12	48	144	432	\	23201
	0.9	2.6	10.4	41.5	124.7	374.1	\	
24	1	3	12	36	144	432	\	24320
	0.9	2.6	10.4	31.2	124.7	374.1	\	
25	1	3	12	36	108	432	\	26124
	0.9	2.6	10.4	31.2	93.5	374.1	\	
26	1	4	12	36	108	432	\	25723
	0.9	3.5	10.4	31.2	93.5	374.1	\	
27	1	4	12	48	144	432	\	24128
	0.9	3.5	10.4	41.5	124.7	374.1	\	
28	1	4	12	36	144	432	\	23027
	0.9	3.5	10.4	31.2	124.7	374.1	\	
29	1	3	9	27	108	576	\	26089
	0.9	2.6	7.8	23.4	93.5	498.8	\	
30	1	3	9	36	108	576	\	25902
	0.9	2.6	7.8	31.2	93.5	498.8	\	
31	1	3	9	36	144	576	\	27231
	0.9	2.6	7.8	31.2	124.7	498.8	\	
32	1	4	16	48	144	432	\	22027
	0.9	3.5	13.9	41.5	124.7	374.1	\	
33	1	3	9	36	144	576	\	23550
	0.9	2.6	7.8	31.2	124.7	498.8	\	
34	1	4	16	48	144	432	\	24528
	0.9	3.5	13.9	41.5	124.7	374.1	\	
35	1	4	16	64	192	576	\	25950
	0.9	3.5	13.9	55.4	166.3	498.8	\	
36	1	4	16	48	192	576	\	25872
	0.9	3.5	13.9	41.5	166.3	498.8	\	
37	1	4	16	48	144	576	\	25848
	0.9	3.5	13.9	41.5	124.7	498.8	\	

代码：

Code: best-height.py

```
# input

target = []

# end

l = []
```

```
while len(l) < 8:    l.append(1)
T = target
le = len(T)
while len(T) < 8:    T.append(1)
T.sort(reverse=True)


T1 = T[0]
T2 = T[1]
T3 = T[2]
T4 = T[3]
T5 = T[4]
T6 = T[5]
T7 = T[6]
T8 = T[7]


for i in range(1, 485):
    a = i // T1
    l[-1] += a
    b = (i - a * T1) // T2
    l[-2] += b
    c = (i - a * T1 - b * T2) // T3
    l[-3] += c
    d = (i - a * T1 - b * T2 - c * T3) // T4
    l[-4] += d
    e = (i - a * T1 - b * T2 - c * T3 - d * T4) // T5
    l[-5] += e
    f = (i - a * T1 - b * T2 - c * T3 - d * T4 - e * T5) // T6
    l[-6] += f
    g = (i - a * T1 - b * T2 - c * T3 - d * T4 - e * T5 - f * T6) // T7
    l[-7] += e
```

```
    h = (i - a * T1 - b * T2 - c * T3 - d * T4 - e * T5 - f * T6 - g * T7)
// T8
    l[-8] += f


for i in range(5):
    print("T" + str(i + 1), ":", l[i])


s = l[0] + l[1] + l[2] + l[3] + l[4] + l[5] + l[6] + l[7]
print("=" * 20)
print("sum:", s)        # sum of all blocks
print("sum2:", s * le)  # sum * number of target
```

Code: loop-best-height3

```
mini = [1000000, 1000000, 1000000]
for k in range(3, 101):
    for j in range(2, 101):
        l = [0, 0, 0, 0, 0]
        T = [1, j, k]
        le = len(T)
        while len(T) < 5:    T.append(1)
        T.sort(reverse=True)


        T1 = T[0]
        T2 = T[1]
        T3 = T[2]
        T4 = T[3]
        T5 = T[4]


        for i in range(1, 485):
```

```
            a = i // T1
            l[-1] += a
            b = (i - a * T1) // T2
            l[-2] += b
            c = (i - a * T1 - b * T2) // T3
            l[-3] += c
            d = (i - a * T1 - b * T2 - c * T3) // T4
            l[-4] += d
            e = (i - a * T1 - b * T2 - c * T3 - d * T4) // T5
            l[-5] += e

        s = l[0] + l[1] + l[2] + l[3] + l[4]
        s2 = s
        if s2 < mini[2]:
            mini[0] = j
            mini[1] = k
            mini[2] = s2
print("=" * 20)
print("j:", mini[0])
print("k:", mini[1])
print("sum:", mini[2])
```