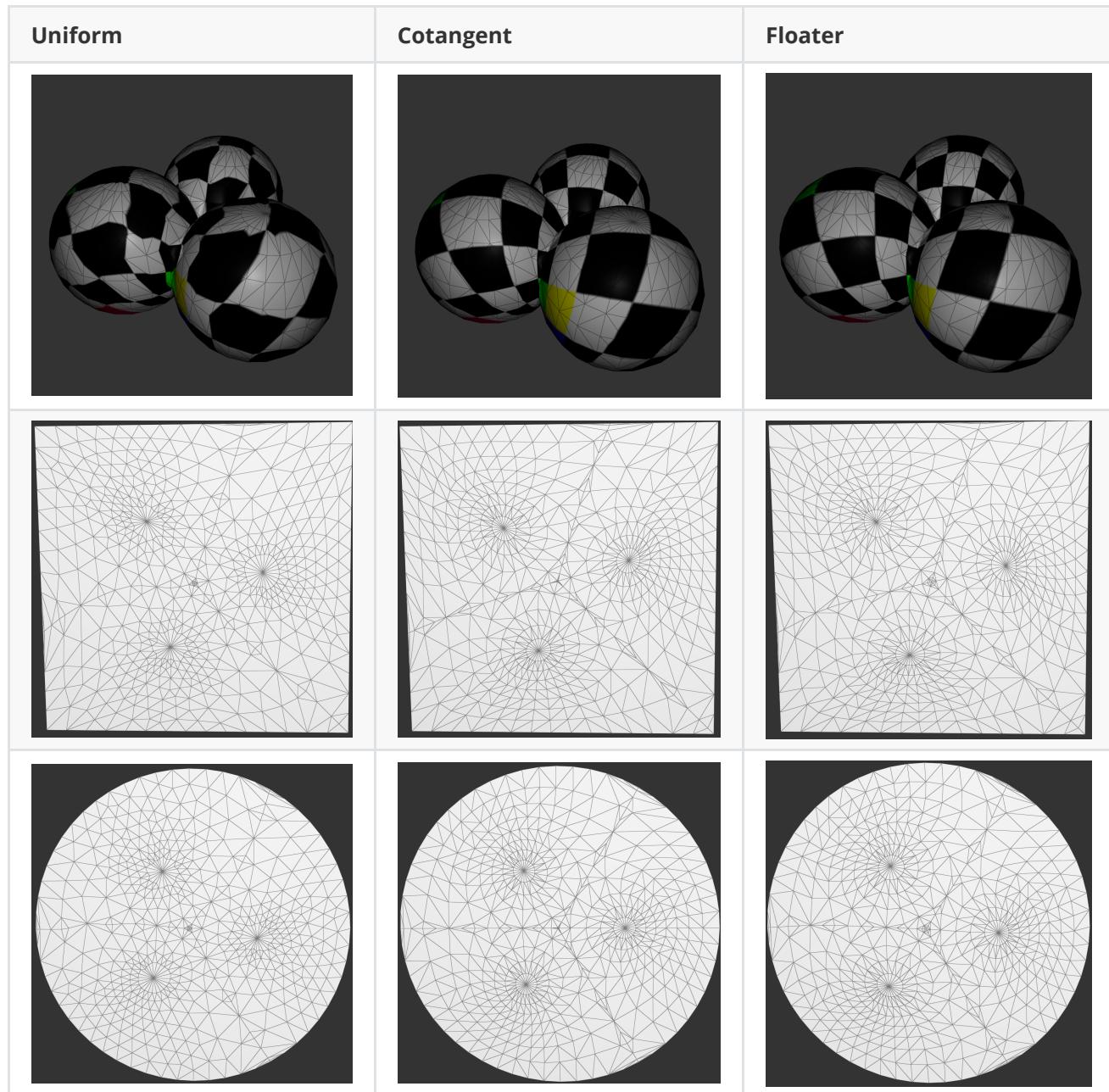


Homework 4 作业报告 by 76-朱雨田

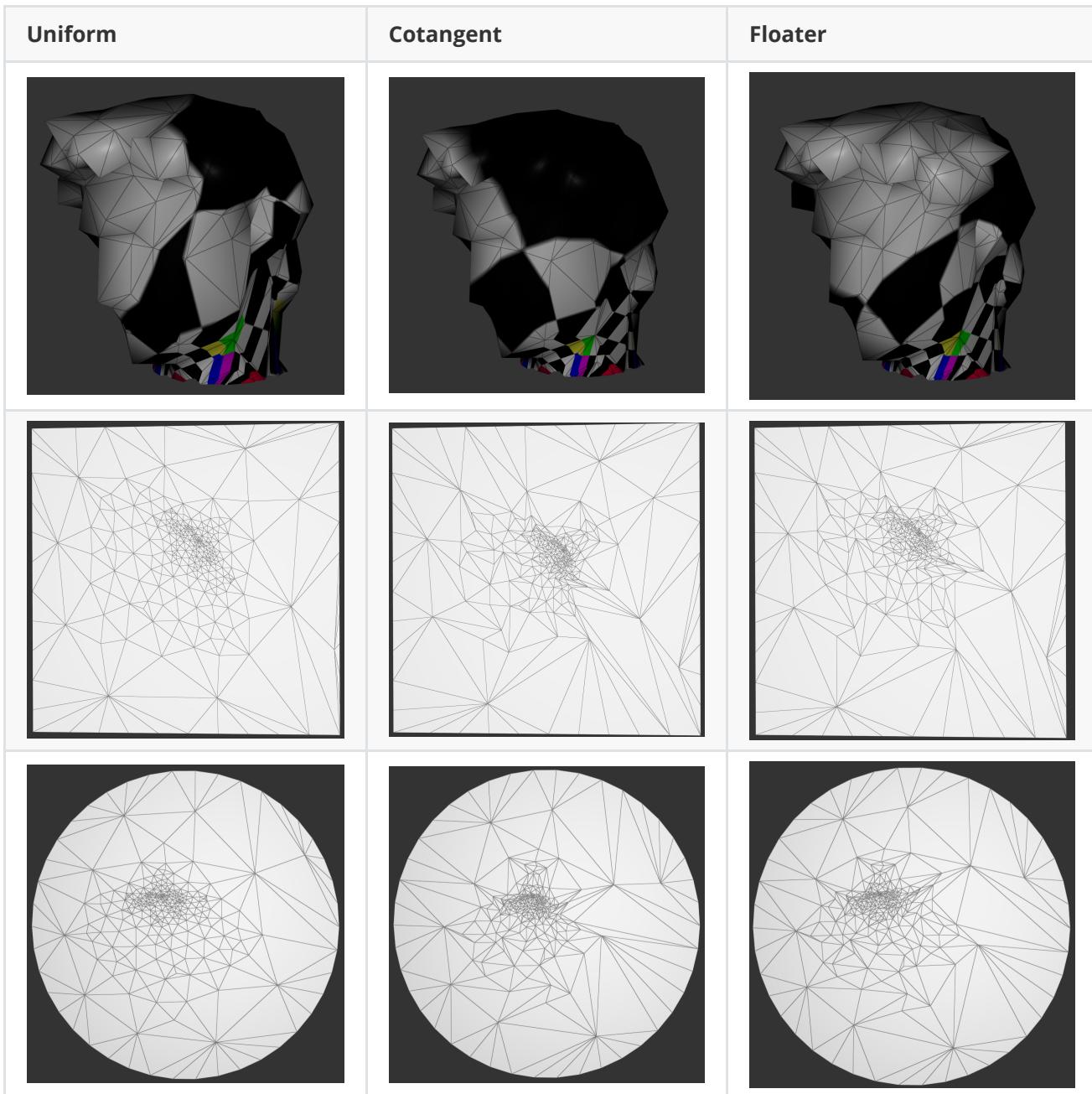
本次作业要求实现求极小曲面以及 Uniform, Cotangent 和 Floater 三种权重的线性曲面参数化算法，并理解半边数据结构和节点化编程思想。从这次作业开始，我们用上前三次作业学到的数学与计算机知识，开始攻克计算机图形学的第一个领域——几何啦！

我完成了以上所有作业要求，并比对了各个参数化方法，研究了本次作业中固定边界曲面参数化算法的局限性。

Balls 模型的效果展示如下。可以注意到 Cotangent 权重和 Floater 权重对原模型中平滑的线条也做了平滑的映射，使得贴上纹理的结果更为平滑。



David 模型的效果如图所示。可以注意到因为颈部尺寸过小而引起的扭曲，使得头部在纹理图中被分配的面积过小。

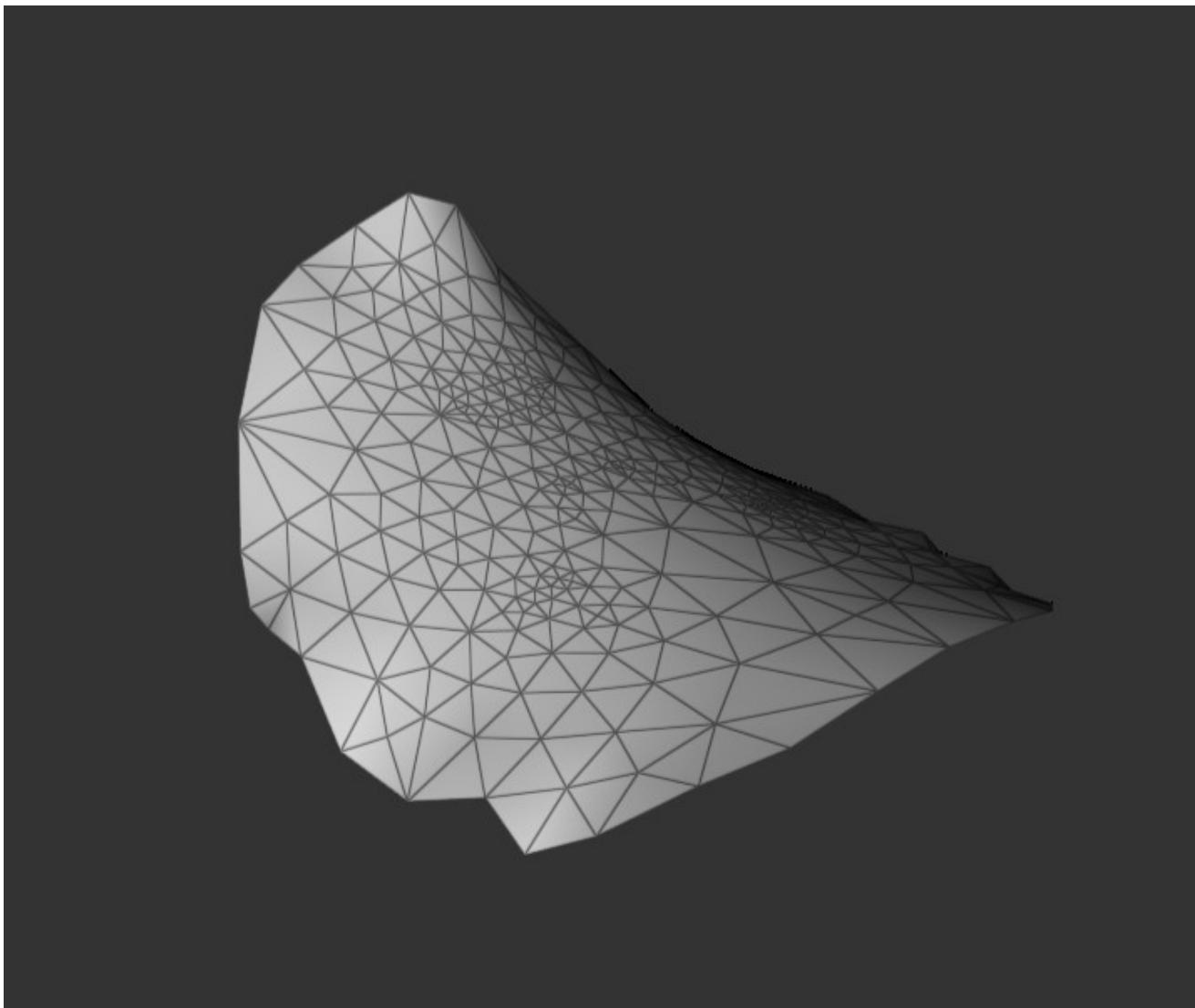


一. 极小曲面

三维曲面上点的 **Laplace 坐标** 定义为其坐标与邻域坐标平均值的差。

极小曲面即除了边界点外，各点 Laplace 坐标均为 0 的曲面，其因为相同边界下面积最小的特性，在建筑学等学科中常常得到应用。

这里只需和 Homework3 Poisson Image Editing 一样，列出带 Dirichlet 边界条件的 Laplace 方程并求解即可。以下是 NefertitiFace 模型的极小曲面展示：



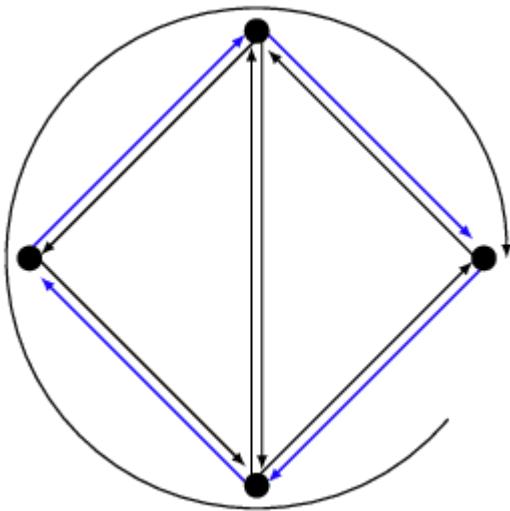
二. Uniform 与 Cotangent 权重曲面参数化

为了将 2D 的纹理映射到 3D 曲面上，以使曲面表面具有色彩、材质等特性，我们需要将 3D 曲面映射到二维空间，给每个顶点赋予一个**纹理坐标**，并将非顶点的位置的纹理坐标进行**插值**，以为曲面上的每个点确定其在二维空间中的精确坐标。这就是**曲面参数化**。

本次作业中的曲面参数化均属于**固定边界参数化算法**，即将边界映射到固定的凸多边形上，再求带权重的 Laplace 方程将曲面映射到平面上。调节映射结果的方法即是调整权重。本次作业中的参数化算法均可以通过解线性稀疏方程组实现，和 Homework 3 相同。这里更需要思考的是关于 OpenMesh 库中半边数据结构的灵活使用。

不要重复造轮子——当我们需要一个“很常用但写起来似乎有点麻烦”的功能时，应当想想库中是否已经实现了相应功能。这里遇到的看起来比较难以实现的两处分别是**按顺时针顺序获取边界点**和 Cotangent 权重中的**按顺时针顺序获取一个点的相邻点**。

如果不考虑库的特性，朴素的按顺序获取边界点的方法是先获取一个边界点，再依次遍历图。这样写起来稍微有点麻烦——事实上，OpenMesh 有一个非常好的特性可以实现这个功能：边界上的半边的 `next()` 仍是在边界上的半边。如图（摘自 [OpenMesh学习笔记6 怎样遍历网格 next_halfedge_handle-CSDN博客](#)）：

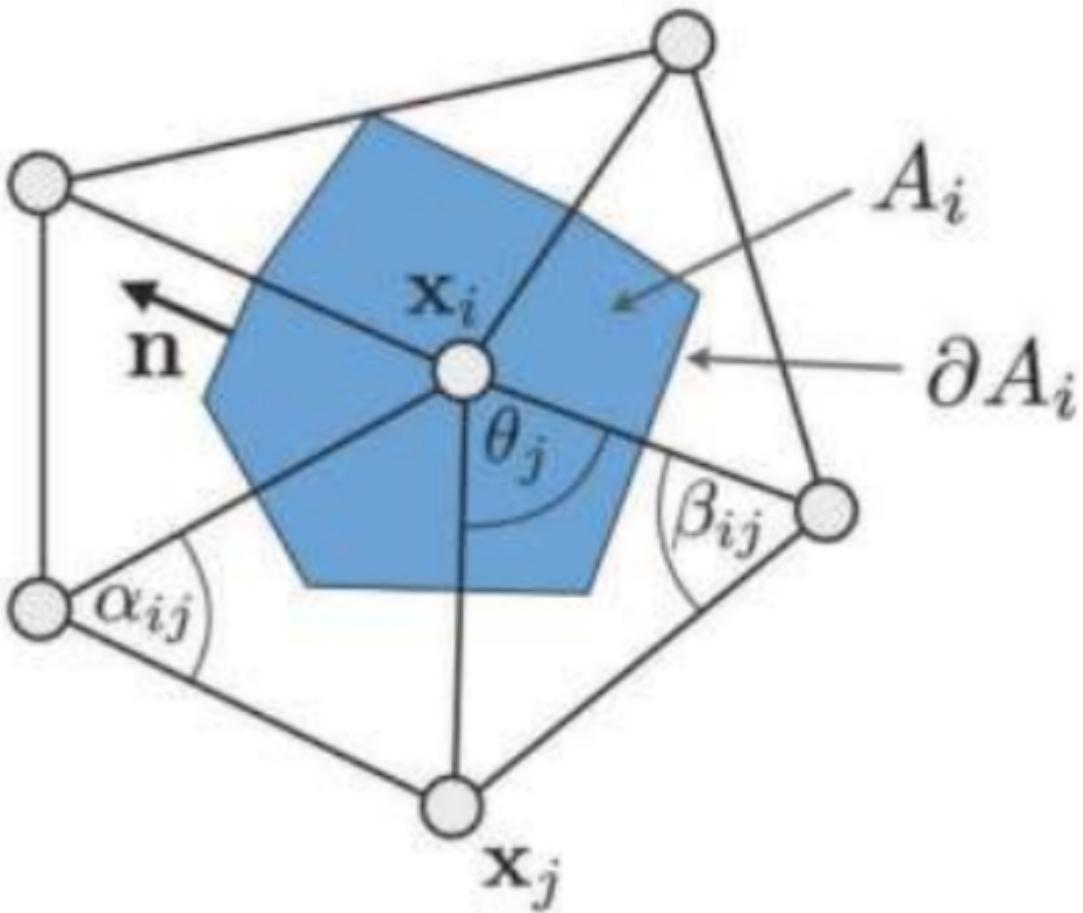


这样，我们可以轻松地按顺序遍历边界，代码如下：

```

1 int n_boundary_vertices = 0;
2 std::vector<OpenMesh::SmartVertexHandle> boundary; // 将所有边界点按顺序存在 vector 中
3 OpenMesh::SmartHalfedgeHandle cur_boundary; // 任意记录一条边界上的半边
4 for (auto he : halfedge_mesh->halfedges())
5 {
6     if (he.is_boundary())
7     {
8         n_boundary_vertices++;
9         cur_boundary = he;
10    }
11 }
12 for (int i = 0; i < n_boundary_vertices; i++)
13 {
14     boundary.push_back(cur_boundary.from());
15     cur_boundary = cur_boundary.next();
16 }
```

比起遍历图按顺序获取边界点，按顺序遍历获取一个点的邻接点写起来要更麻烦。OpenMesh 也为我们提供了顺时针的迭代器 `outgoing_halfedges_cw()`，可以实现按顺序的 for 循环。（如果不是查阅资料，实在很难联想到 cw 居然是 **clockwise** 的意思，从而忽略这个好用的迭代器…… C# 设计成所有属性方法名均用英文全称是不无道理的）



```

1 // vertex 是要取邻域的中心顶点
2 std::vector<OpenMesh::SmartVertexHandle> neighbor_list;
3 for (auto outedge : vertex.outgoing_halfedges_cw())
4 {
5     neighbor_list.push_back(outedge.to());
6 }

```

从报告开头处的表格对比可发现，Cotangent 权重参数化设置的纹理明显比 Uniform 权重有更少的扭曲，因为 Cotangent 权重保留了原本平滑（共面？）的边的平滑性。这意味着固定曲面参数化算法中的权重有着至关重要的作用。

实现 Cotangent 权重参数化的时候注意求 Laplace 坐标的时候要将**原来的 Mesh**传入，而不应把设置了固定边界的 Mesh 传入，否则在边界附近会因为边长、角度发生变化而产生巨大的变形。

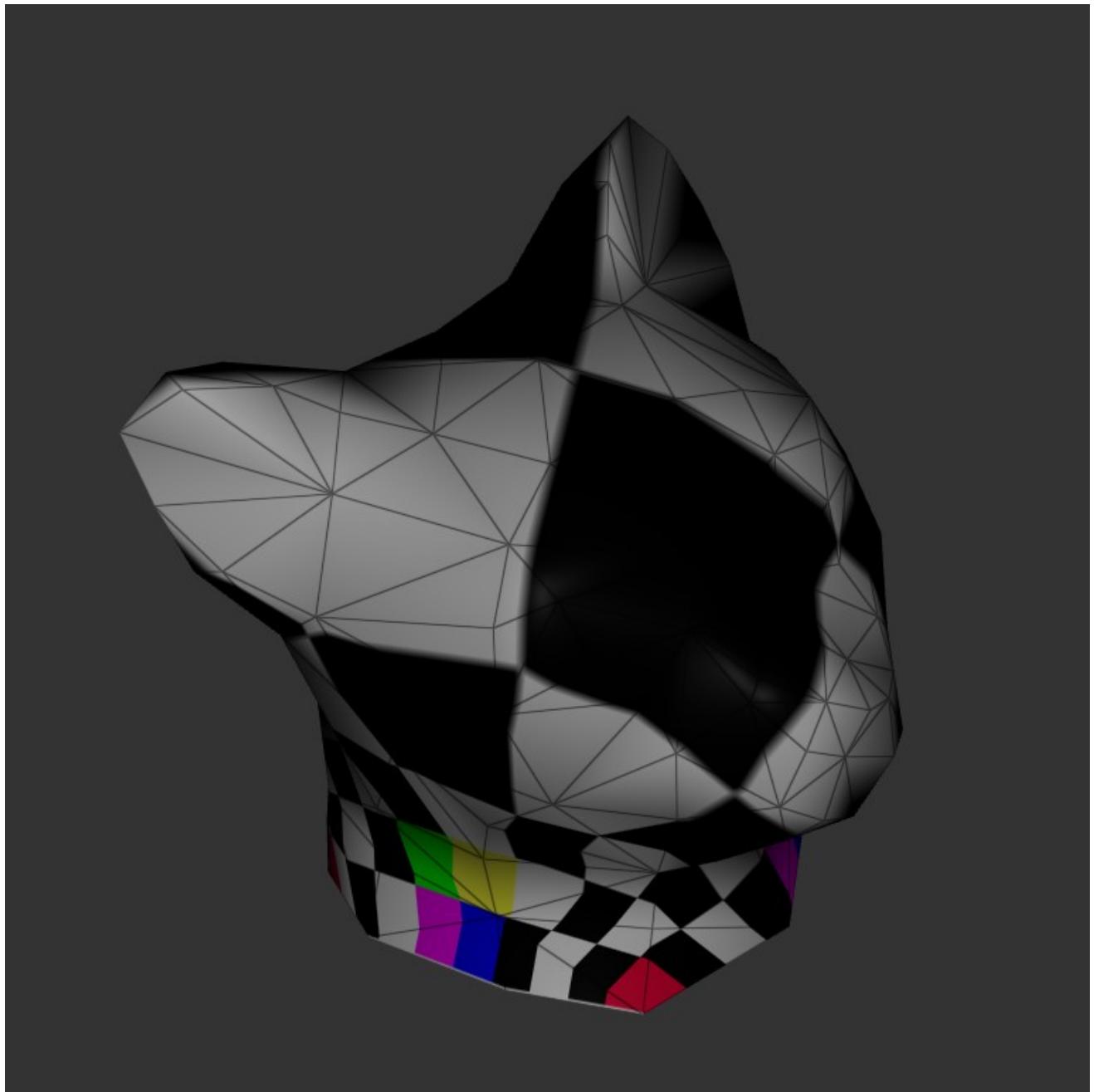
向正方形边界映射时会出现正方形顶点处空缺的情况。若是均匀将边界映射到线段上，再将线段“折成四份”生成正方形，只需将顶点附近距顶点距离小于步长一半的点放在正方形顶点上即可。注意这里需要用 `double` 类型。

`float` 类型的浮点精度误差会造成 David 模型映射后的正方形缺角（调了好久才发现...）

```
1 double step = 4.0 / n_boundary_vertices;
2 for (int i = 0; i < n_boundary_vertices; i++) {
3     double t1 = i * step;
4     int edge = (int)floor(t1);
5     double t = t1 - floor(t1);
6     if (t <= step / 2)
7         t = 0;
8     if (1 - t < step / 2)
9         t = 1;
10    // DO SOMETHING...
11 }
```

因为这两种参数化算法是针对固定边界的，仍可以用于求极小曲面，因此本次作业中的新节点名仍被命名为 Minimum Surface (Uniform Weight) 和 Minimum Surface (Cotangent Weight)。

下图是使用了 Cotangent Weight 的 CatHead 模型纹理映射：



三. Floater 权重参数化

Floater 权重参数化是一种保形映射。以下是百度百科对保形映射的介绍：

所谓保形映射是指满足以下两个条件的映射：①过一定点的曲线的正向切线到其象曲线上对应点的正向切线的转角是一个与曲线的选择无关的常数，称其为映射在定点的转动角度。②过一定点的象曲线上一动点到定点的距离与原象曲线上对应点的距离之比，当动点沿曲线趋向定点时的极限为一与曲线的选取无关的常数，称其为映射在定点的伸缩率。上述性质①有一种等价的形式：①' 过定点的任意两条曲线经映射后其转角的大小及方向均不变，形象地称这一性质为同向保角性，①'与②一起表明在一定点附近的一个小三角形，与其象“三角形”（一般是曲边三角形）近似地同向相似，称其为保形映射。

其主要有以下步骤：

1. 对于一个顶点，将其邻域构成的三维多边形**保持角度比例**映射到平面上的一个二维多边形，并将该顶点作为原点（如图， $\angle \mathbf{p}_1 \mathbf{p} \mathbf{p}_2 = \frac{2\pi \angle \mathbf{x}_{j_1} \mathbf{x}_i \mathbf{x}_{j_2}}{\sum \angle \mathbf{x}_{j_k} \mathbf{x}_i \mathbf{x}_{j_{k+1}}} ;$ ）；
2. 对每个邻接节点，在二维多边形上找一条**相对的边**，使得原点在该点与该边两个顶点的连线构成的三角形**内部**（如下图 $\triangle \mathbf{p}_{r(l)} \mathbf{p}_l \mathbf{p}_{r(l)+1}$ ，存在唯一的三角形使得对于每个 \mathbf{p}_l ，有原点 \mathbf{p} 在其内部）。不难想到这样的边若存在，则是唯一的。不存在的情况只有存在一个正好相对的点，此时将其相邻的边作为这样的边即可。
3. 为三角形的三个顶点权重加上它们各自的重心坐标（如对于 \mathbf{p}_l ，为其权重加上 $\frac{S_{\triangle \mathbf{p} \mathbf{p}_l \mathbf{p}_{r(l)+1}}}{S_{\triangle \mathbf{p}_{r(l)} \mathbf{p}_l \mathbf{p}_{r(l)+1}}}$ ）。

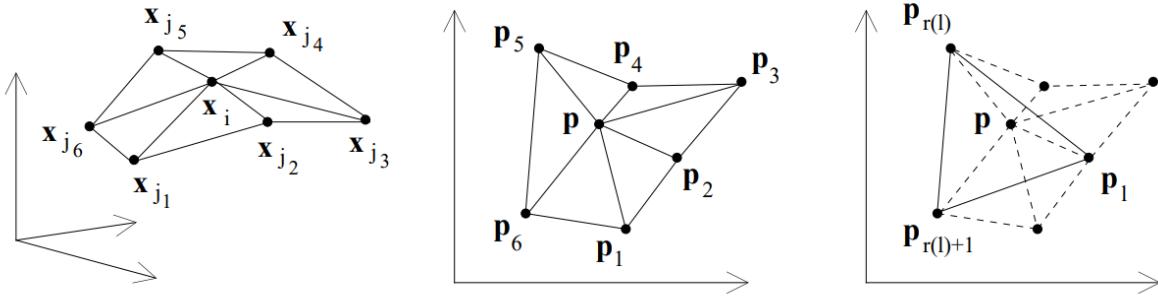


Fig. 5. The subtriangulation $S_i(G_i, \mathbf{X}_i)$ and a local parametrization \mathcal{P}_i .

因为三个顶点以重心坐标加权求和后得到的即是原点，因此将二维多边形上所有顶点加权求和后仍能得到顶点。这就是该权重得以做到保形映射的理论基础。

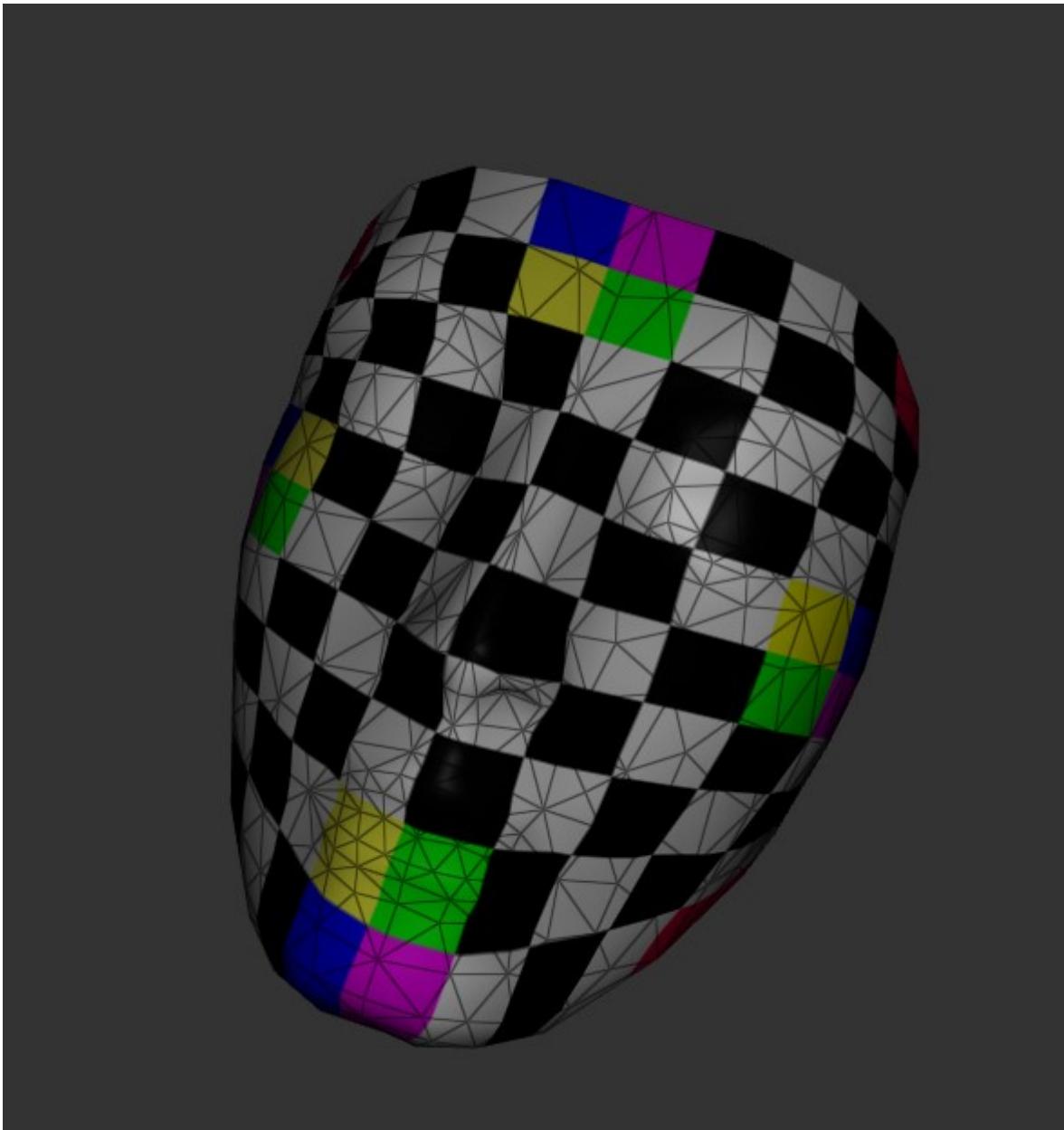
其中步骤 2 的实现比较具有技巧性——当 \mathbf{p}_l 逆时针行进的时候，其对应的 $\mathbf{p}_{r(l)}$ 一定也会向逆时针方向行进，因此在每次 \mathbf{p}_l 更新后用 `while` 循环找到第一个满足条件的 $\mathbf{p}_{r(l)}$ 即可。判断原点是否在三角形内部只需用两个叉乘（行列式）判断，面积也可以用叉乘求解。代码如下：

```
1 // 这里已经将邻域上每个点的二维坐标有序地存进了 coord2d 数组中。
2 // back_ind 是相对边上一个端点。
3 int back_ind = 0;
4 for (int i = 0; i < n_neighbor; i++)
5 {
6     // 第一个条件是防止相对边的端点就是当前顶点，这种情况也是满足叉乘条件的。
7     // 为了保证原点在三角形内部，这个叉乘的值必须 <= 0
8     // 因为凸多边形的边界是环状的，因此对 back_ind 取模就能很好地体现环的性质。
9     while (back_ind % n_neighbor == i ||
10            cross2d(coord2d[i], coord2d[i] - coord2d[back_ind % n_neighbor]) *
```

```
11         cross2d(coord2d[i], coord2d[i] - coord2d[(back_ind + 1) % n_neighbor]) >
12     0)
13     back_ind++;
14 }  
// DO SOMETHING...
```

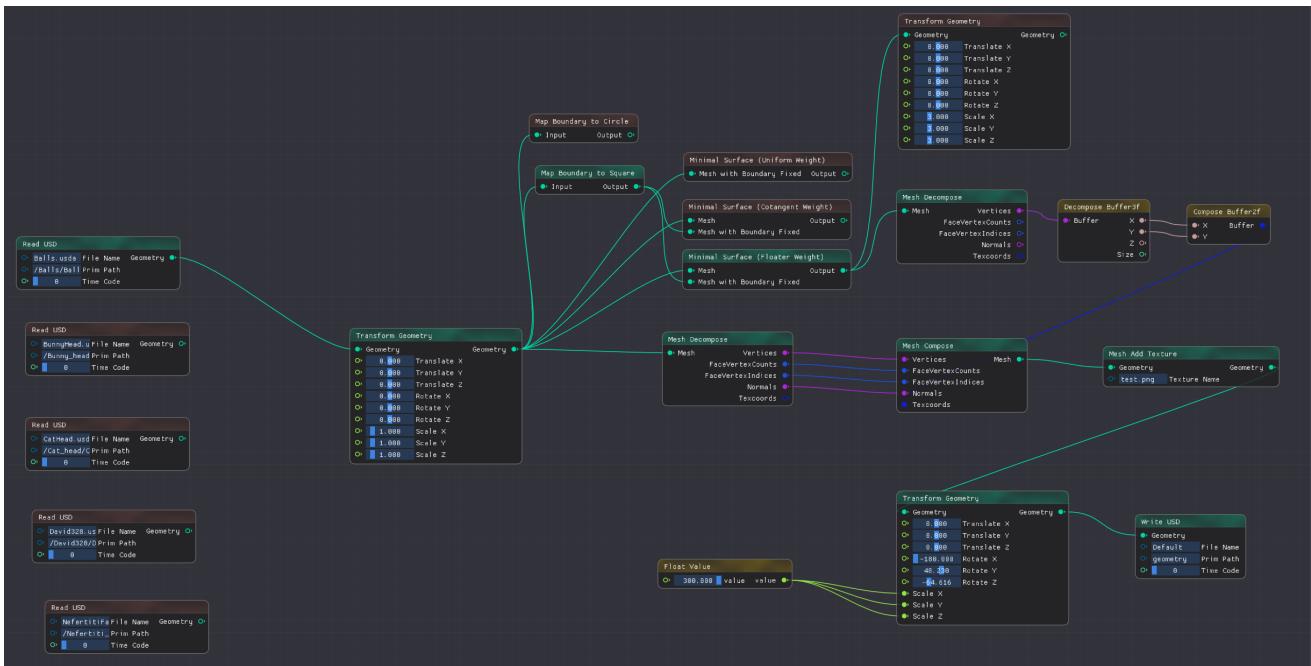
注意步骤 3 中的权重一定要除下总面积，否则即使加权求和仍在原点，也不再保证保形映射的性质，会生成扭曲的映射结果。

以下是使用了 Floater 模型的 Nefertiti Face 纹理映射：



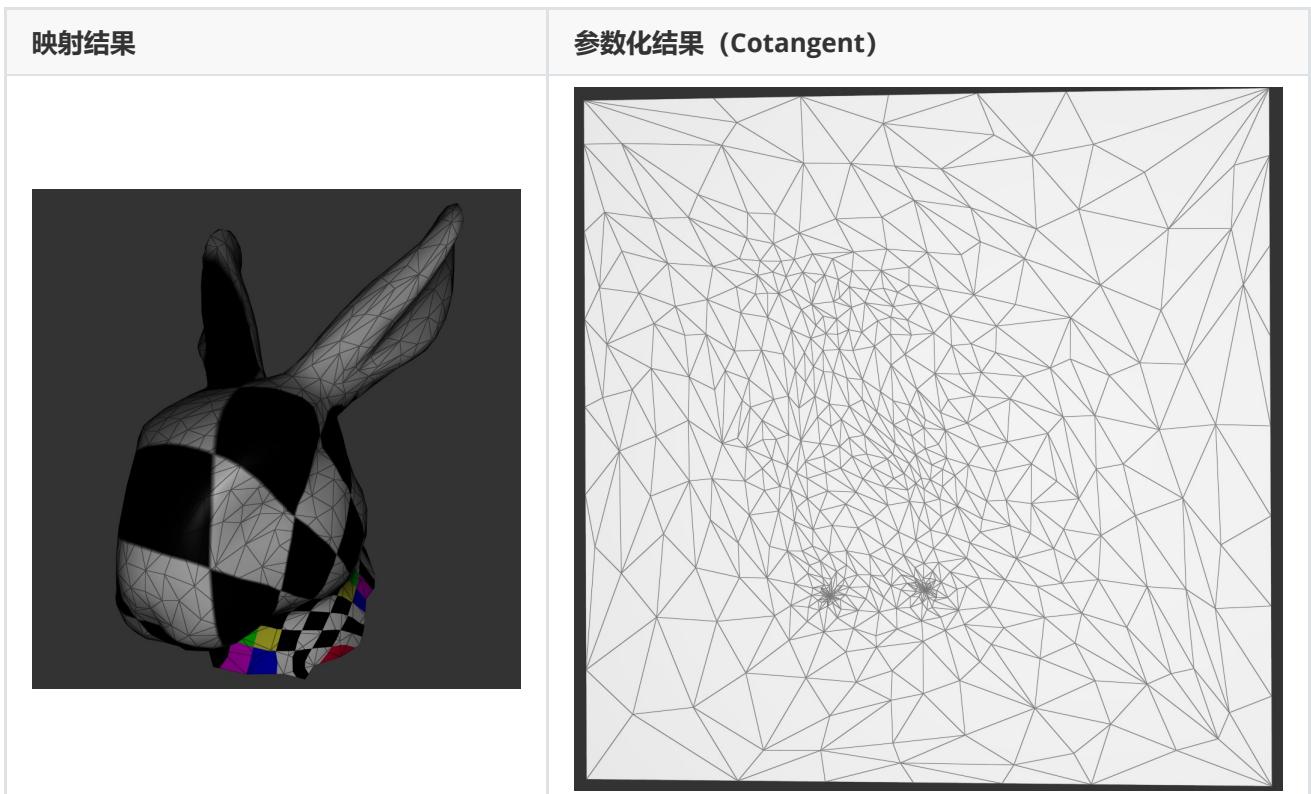
四. 节点化编程

节点化编程在 Unity 的 Shader Graph 和 Blender 的几何节点中已经有了很成熟的应用。图形学的几何处理、图像处理常常并不需要非常复杂的条件逻辑，其程序模式更接近节点化编程中的 DAG 结构。节点化编程也简化了阅读程序的难度，还可以展示每个节点中的图像效果，更有利于调试与非程序员的参与。以下是本次作业用到的节点图：



五. 固定边界参数化算法的局限性展示

即使以凸多边形为边界的曲面参数化能保证三角形不被翻转，其对曲面形状的扭曲仍是很大的。如报告开头的的 David (较窄的颈部使得颈部附近被分配了很大的面积，而头部分配的面积却很小)，以及下图中的 Bunny Head (两只耳朵被分配了很小的面积)



就算权重调整地再完美，我们认为这样的映射效果仍然是不尽人意的。如果要保证更小的扭曲，我们需要对曲面做出更好的切割——但切割成不规则的曲面后再映射到凸多边形边界，必然会造成曲面的更大扭曲。因此，研究**非固定边界的曲面参数化**算法显示了其重要的价值。

心得体会

这次配置框架的时候遇到了不少问题，不过群里的助教非常热心地回答了所有问题，列出如下：

1. Visual Studio 跑 Current Document 时出错——需要只构建 engine_test。是我习题课听漏了！
2. 节点中的拖动条用 `ctrl+左键单击` 即可手动键入数值。
3. 自定义节点的 namespace 必须要遵循大括号前有空格、大括号不换行的格式，以被 python 的匹配机制正确识别。助教承诺了优化这一点，非常给力。
4. 兔头模型的尺寸非常小，需要放大 100 倍才能和达到其他模型一样的大小。
5. 将两个 Transform Geometry 节点相连后接入 Write USD 程序会直接崩溃，并且报错。目前已经修复。
6. 加入第一人称相机。

当时做 2020 年的作业 4 时也是配了一晚上框架（好崩溃好崩溃），也是从那时候开始放弃单打独斗开始联系助教解决框架问题的。这次有了群里的学习环境后，省心了不少。

这次写代码时给 Visual Studio 装了个 Vim 插件。虽然还用不习惯但也慢慢爽起来了。

Floater 的论文仍然是借助 ChatGPT 阅读的，只读了保形映射的一节。这篇论文读起来难度还是不大的，特别是有了 ChatGPT 的辅助之后只需要十几分钟就能看明白。

作业难度开始上升了，希望以后也能保持 Optional 全收集，通关整个图形学课程吧。