HW4

HW4最难的地方不是算法的实现，而是处理各种奇怪的`bug`，于是我们先来看看此次作业遇见的问题，或许能够帮助完善实验文档

神秘的问题

怎么是`segmentation fault`？

由于我们的文件结构是这样的

- Framework3D
  - Framework3D
    - Binaries
      - Release
        - USTC_CG_polytype_test.exe

所以，当我们`vscode`的工作区的根目录是最顶端的`Framework3D`时，我们的终端也是工作区的根目录，这是如果要运行exe文件，我们居然要先`cd`到里面的`Framework3D`目录，再运行`./Binaries/Release/USTC_CG_polytype_test.exe`，否则可能会提醒`segmentation fault`

原因猜测

可能是因为vscode一开始打开的工作区目录是里面的`Framework3D`，编译以及运行都得在原工作区目录下

后续处理

在最顶端的`Framework3D`，删掉所有的`build`，重新`cmake`就好了

遗留下的问题

每次`cmake`和编译相关的问题，在没有出现什么明确的代码错误时，解决方法似乎总是指向`rm -rf ./build` and `cmake ..`，有点像出问题，就重启电脑，有没有其他更好的解决办法呢？因为项目比较大之后，每次完全的`build`一个项目的时间还挺长的，能不能删除特定的文件，而不是删除整个`build`？

Link error：怎么找不到`shaderc_combinedd.Lib`啊？

我的目录下怎么只有`shaderc_combinedd.Lib`？神秘的原因居然是`VulKan`选下载组件没选全，这谁一开始就想得到要全部下载

当然，还有两种黑魔法

- 重命名当前的`Lib`文件，把末尾的`dd`删掉一个
- 或者换成`Release`配置

怎么是编译器堆栈空间不足啊？

`Google`一下，说是`MSVC`默认用`4G`内存去编译（好像是这意思），可以用命令解除限制

好的，查看`cmake`原命令，好家伙已经解开限制了，再次编译，使用`任务管理器`查看运行时内存，好家伙，内存占用直逼`16G`

这下看懂了，编译时不要打开浏览器看视频，最好只开启`vscode`

终于编译好了，怎么不支持显卡？

这就是用`intel`的福报吗？ 😄

算法的实现

先使用`boundary_mapping`，再使用`Laplacian_min_surf`

Circle Boundary Mapping

```
auto input = params.get_input<Geometry>("Input");
auto halfedge_mesh = operand_to_openmesh(&input);
```

考虑单个边界的简单情况

随便找一个边界`half_edge`

```
    for (auto he_it = halfedge_mesh->halfedges_begin(); he_it != halfedge_mesh->halfedges_end(); ++he_it)
    {
        if (halfedge_mesh->is_boundary(*he_it))
        {
            start_he = *he_it;
            break;
        }
    }
```

把边界顶点`vertex`存下来

```
std::vector<OpenMesh::VertexHandle> boundary_vertices;
OpenMesh::HalfedgeHandle current_he = start_he;

    do

    {

        OpenMesh::VertexHandle vh = halfedge_mesh->from_vertex_handle(current_he);

        boundary_vertices.emplace_back(vh);

        current_he = halfedge_mesh->next_halfedge_handle(current_he);

    } while (current_he != start_he);
```

设置他们的`3D`位置

```
    for (int i = 0; i < num_boundary_vertices; ++i)

    {

        double angle = 2.0 * M_PI * static_cast<double>(i) / num_boundary_vertices;

        double x = 0.5 * cos(angle);

        double y = 0.5 * sin(angle);

        halfedge_mesh->point(boundary_vertices[i]) = OpenMesh::Vec3d(x, y, 0.0);

    }
```

转化格式

```
auto geometry = openmesh_to_operand(halfedge_mesh.get());
params.set_output("Output", std::move(*geometry));
```

Square_boundary_mapping

逻辑是类似的，有些代码处理的还挺不错

```
double total_length = 0.0;
std::vector<double> cumulative_lengths;
int vertices_per_side = num_boundary_vertices / 4;
int remainder = num_boundary_vertices % 4;

std::vector<int> side_counts(4, vertices_per_side);
for (int i = 0; i < remainder; ++i)
    {
        side_counts[i]++;
    }
```

例子：

假设 total_vertices = 10。

1. **计算基础数量和余数：**
   - vertices_per_side = 10 / 4 = 2
   - remainder = 10 % 4 = 2
2. **执行代码片段：**

```
    std::vector<int> side_counts(4, 2); // side_counts 初始化为 {2, 2, 2, 2}

    for (int i = 0; i < 2; ++i) // 循环 2 次 (remainder = 2)

        循环 1: i = 0, side_counts[0]++; // side_counts 变为 {3, 2, 2, 2}

        循环 2: i = 1, side_counts[1]++; // side_counts 变为 {3, 3, 2, 2}
```

其中一条边的顶点的分配逻辑

```
    // Bottom edge: (x, y) = (t, 0)

    for (int i = 0; i < side_counts[0]; ++i)

    {

        double t = static_cast<double>(i) / (side_counts[0] - 1);

        halfedge_mesh->point(boundary_vertices[vertex_index++]) = OpenMesh::Vec3d(t, 0.0, 0.0);

    }
```

Min_surf

分开处理边界点和内部点，边界点书需要保持位置不变（边界条件），内部点的位置需要求解

需要对顶点建立一个`index_map`，我们后续需要填充一个`Laplacian Matrix`

```
        std::vector<int> boundary_vertices;
        std::vector<int> internal_vertices;
        std::map<typename OpenMesh::PolyMesh_ArrayKernelT<>::VertexHandle, int> vertex_indices;

        int index = 0;
        for (auto v_it = halfedge_mesh->vertices_begin(); v_it != halfedge_mesh->vertices_end(); ++v_it)
        {
            vertex_indices[*v_it] = index++;
            if (halfedge_mesh->is_boundary(*v_it))
            {               boundary_vertices.emplace_back(vertex_indices[*v_it]);
            }
            else
```

```
            {               internal_vertices.emplace_back(vertex_indices[*v_it]);
            }
        }
```

遍历`halfedge-mesh`上的每一个顶点，通过`index_map`找对应的`index`，边界点好说，只用

```
triplets.emplace_back(T(i, i, 1.0));
auto point = halfedge_mesh->point(*v_it);
b_x(i) = point[0];
b_y(i) = point[1];
b_z(i) = point[2]; //保持位置不变
```

非边界点使用`uniform weight`，对每一个非边界顶点，要查看它的邻居，来填补它对应的那行数据

```
double weight_sum = 0.0;

for (auto vv_it = halfedge_mesh->vv_iter(*v_it); vv_it.is_valid(); ++vv_it)
{
    int j = vertex_indices[*vv_it];
    triplets.emplace_back(T(i, j, -1.0));
    weight_sum += 1.0;
}
triplets.emplace_back(T(i, i, weight_sum));
```

Cotangent Weights

使用边界固定前的`mesh`来计算权重，计算权重的逻辑根据下面的图像

The main idea behind this is to construct and manipulate a *delta coordinates* for each mesh. A delta coordinate for vertex **i** is defined as follows:

$$\vec{\delta_i} = \vec{v_i} - \frac{\sum_{j \epsilon N(i)} w_{ij} \vec{v_j}}{\sum_{j \epsilon N(i) w_{ij}}}$$

Another common weighting scheme involes the use of "cotangent weights" between the angles in this figure:

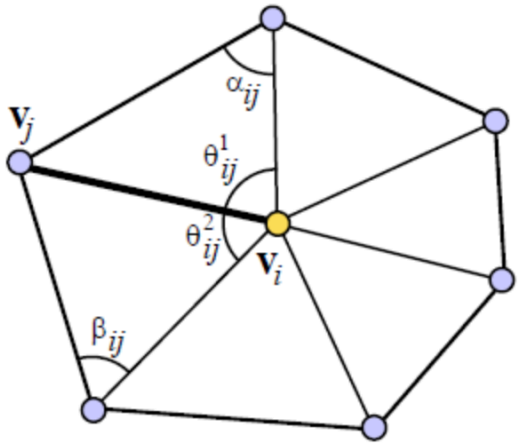


Figure 2: *The angles used in the cotangent weights and the mean-value coordinates formulae for edge* (i, j).
(Picture Courtesy of Olga Sorkine's STAR Report)

我们参考这里的内容, Laplacian Mesh Editing

计算权重的实现代码如下

```cpp
namespace OpenMeshUtils
{

    template <typename MeshT>
    解释代码 | 注释代码 | 生成单测 | ×
    double cotangent_weight(const MeshT &mesh,
                            typename MeshT::HalfedgeHandle he)
    {

        if (!mesh.is_valid_handle(he))
            return 0.0;

        // 获取三个顶点坐标
        const auto v0 = mesh.point(mesh.from_vertex_handle(he));
        const auto v1 = mesh.point(mesh.to_vertex_handle(he));
        // 正确获取对面顶点
        const auto next_he = mesh.next_halfedge_handle(he);
        if (!mesh.is_valid_handle(next_he))
            return 0.0;

        const auto v2 = mesh.point(mesh.to_vertex_handle(next_he));

        // 计算两个边向量
        OpenMesh::Vec3f e1, e2;
        e1 = v0 - v2;
        e2 = v1 - v2;

        // 计算点积和叉乘模长
        const auto dot = OpenMesh::dot(e1, e2);
        const auto cross = OpenMesh::cross(e1, e2).norm();
        const auto cond_max = 1e6;

        // 防止除零
        return (cross < 1e-8) ? 1e6 : (dot / cross);
    }

} // namespace OpenMeshUtils
```

遗留的问题

由于 `cotangent weights` 需要 `mesh` 边界固定前的 `mesh` 来计算，所以我们自然的想法就是：

- 单独抽象出一个节点，输入是原始的 `mesh`，输出是对应的 `Laplacian Matrix (cotangent weights)`
- 再将这个节点的输出，作为 `node_min_surf` 输入的一部分

实践中发现有两个问题

Segmentation fault again

下面是我们 `node_cotangent_weights.cpp` 的最后输出的代码

```
Eigen::SparseMatrix<double> L(n, n);
L.setFromTriplets(triplets.begin(), triplets.end());
params.set_output("Cotangent_weights", L);
```

简单的链接会出现 `Segmentation fault`

怀疑是这行出现的问题

```
params.set_output("Cotangent_weights", L);
```

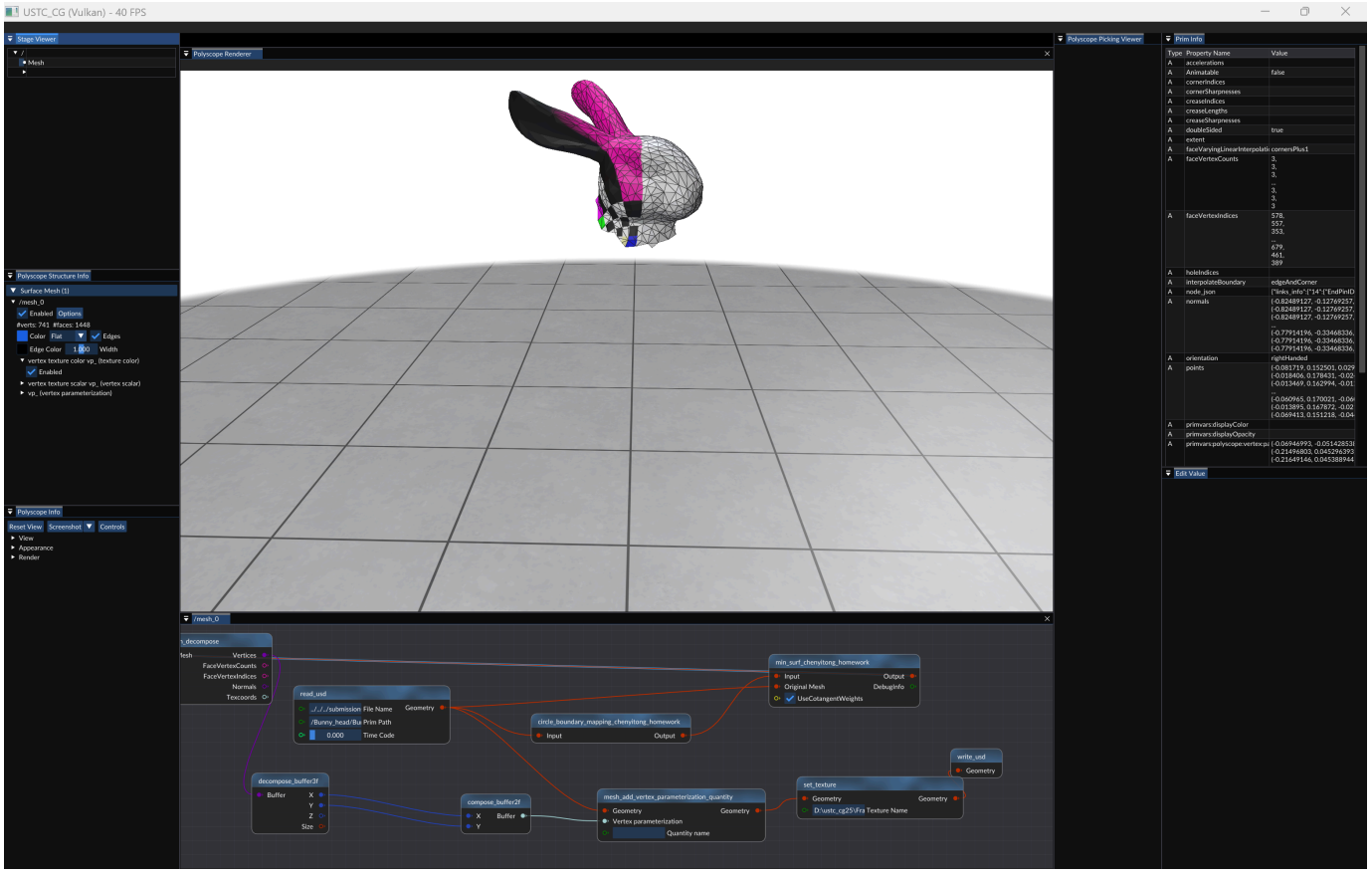
不太清楚节点的输入如何恰当的设置

在 `node_min_surf.cpp` 文件中，节点输入的设置为

```
b.add_input<Geometry>("Input");
b.add_input<Geometry>("Original Mesh");
b.add_input<bool>("UseCotangentWeights").default_val(false);
```

发现不输入 `Original Mesh`，这个节点并不会开始计算

结果展示



将 `circle_boundary_mapping` 替换成 `square` 的，效果更好