作业 08：基于C++的Grasshopper插件制作

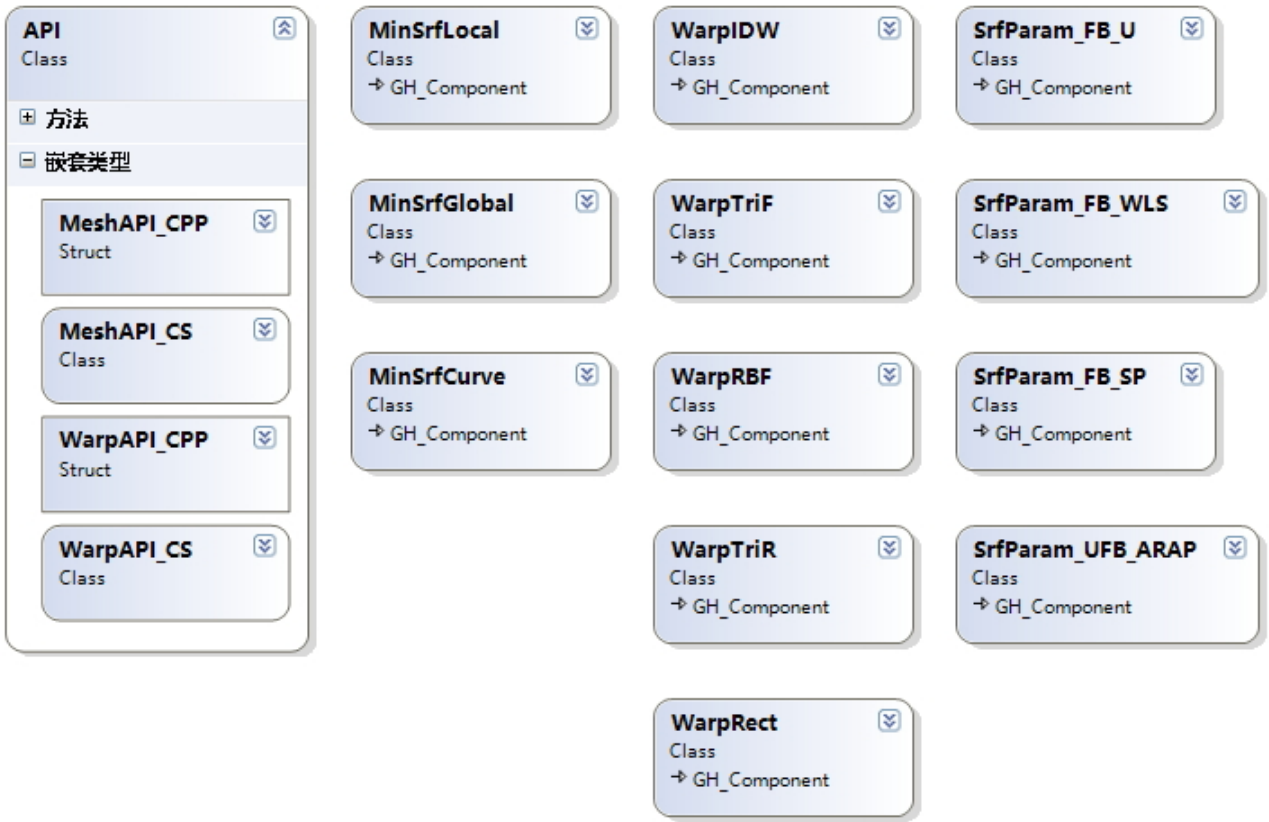
> 85
>
> 庄涛
>
> PB15111679
>
> 2018/04/20

1. 作业要求

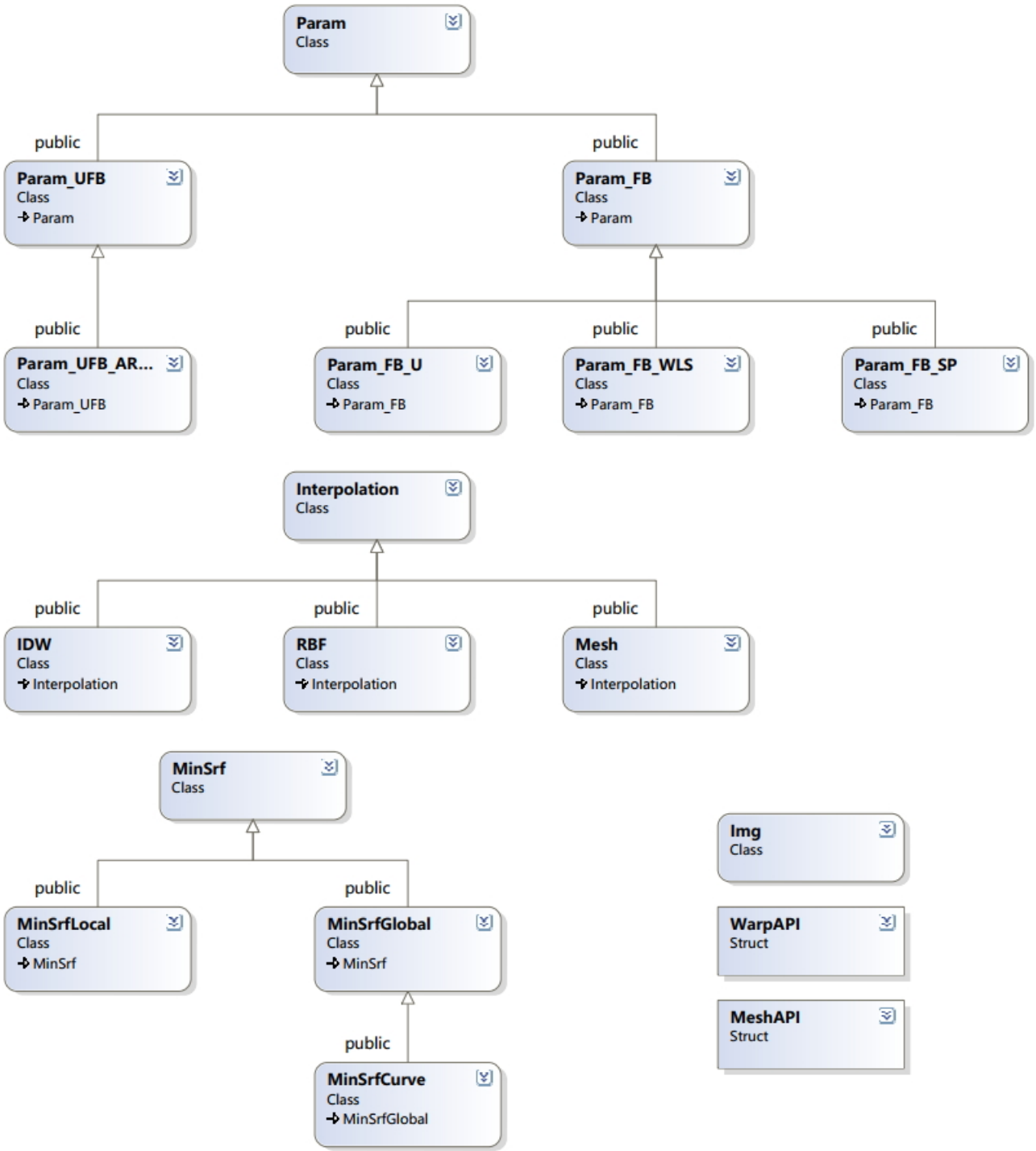
- 制作`Grasshoper`插件族，集成**作业2-5**所实现的功能
- 在`C# Script`电池上使用`Pinvoke`调用已写好的`C++(DLL)`代码

2. 类图

2.1 C#



2.2 C++


Param
Class
public
Param_UFB
Class
Param
public
Param_FB
Class
Param
public
Param_UFB_AR...
Class
Param_UFB
public
Param_FB_U
Class
Param_FB
public
Param_FB_WLS
Class
Param_FB
public
Param_FB_SP
Class
Param_FB
Interpolation
Class
public
IDW
Class
Interpolation
public
RBF
Class
Interpolation
public
Mesh
Class
Interpolation
MinSrf
Class
public
MinSrfLocal
Class
MinSrf
public
MinSrfGlobal
Class
MinSrf
public
MinSrfCurve
Class
MinSrfGlobal
Img
Class
WarpAPI
Struct
MeshAPI
Struct

3. 接口结构体

总共两大类结构体`WarpAPI`和`MeshAPI`。

每类结构体分成三种，`C#`中两种，`C++`中一种。具体如下表

接口	涉及	C#	C++
`Warp`(图像)	`作业2`和`作业3`	`WarpAPI_CS` 和 `WarpAPI_CPP`	`WarpAPI`
`Mesh`(网格)	`作业4`和`作业5`	`MeshAPI_CS` 和 `MeshAPI_CPP`	`MeshAPI`

在`C#`中，`*_CPP`用于和`*_CS`链接，即`*_CPP`中的指针变量与`*_CS`中的数组变量链接。

而`*_CS`中会有一些函数用来跟`grassshoper`中的对象（如`mesh`，`curve`等）进行相互转换。

3.1 Warp

Warp相关的接口是用于图像处理的。

3.1.1 WarpAPI_CS

```
public class WarpAPI_CS
{
    // image
    public int[] R;
    public int[] B;
    public int[] G;
    public int width;
    public int height;
    // imput
    public int[] px;
    public int[] py;
    public int[] fpx;
    public int[] fpy;
    public int n;
    //method
    public WarpAPI_CS(){/*...*/}
    public void loadFrom(ref Mesh mesh, ref List<Point3d> p, ref List<Point3d> fp){/*...*/}
    public void convertTo(ref Mesh mesh){/*...*/}
}
```

`px`和`py`是输入的原点，`fpx`和`fpy`是相应的向量。`loadFrom`和`convertTo`用来和`mesh`交互。

3.1.2 WarpAPI_CPP

```
public unsafe struct WarpAPI_CPP
{
    // image
    public int* R;
    public int* B;
    public int* G;
    public int width;
    public int height;
    //input
    public int* px;
    public int* py;
    public int* fpx;
    public int* fpy;
    public int n;
    //method
    public void linkWith(ref WarpAPI_CS api_cs){/*...*/}
}
```

相应`WarpAPI_CS`的结构，作为C#调用的C++函数的参数的结构。`linkWith`用来将函数指针和`WarpAPI_CS`里的数组对应，使得C++函数能对其直接赋值。

3.1.3 WarpAPI

```
struct WarpAPI{
    // image
    int * R;
    int * G;
    int * B;
    int width;
    int height;
    // input
    int * px;
    int * py;
    int * fpx;
    int * fpy;
    int n;
    // method
    void convertTo(vector<ND_Point> & p, vector<ND_Point> & fp, Img & img);
    void loadFrom(Img & img);
};
```

这是`C++`里边用的结构，`convertTo`用来跟`C++函数`对接。

3.2 MeshAPI

Mesh相关的API结构跟Warp的类似。

3.2.1 MeshAPI_CS

```
public class MeshAPI_CS
{
    public int VerticesNumber;
    public int FaceNumber;
    public int TexCoordNumber;

    public float[] VerticesX;
    public float[] VerticesY;
    public float[] VerticesZ;

    public int[] FaceA;
    public int[] FaceB;
    public int[] FaceC;

    public float[] TexCoordX;
    public float[] TexCoordY;

    public MeshAPI_CS(){/*...*/}
    public void loadFrom(ref Mesh mesh){/*...*/}
    public void loadFrom(ref Curve curve, int PointNum){/*...*/}
    public void loadFrom(ref MeshAPI_CPP api_cpp){/*...*/}
    public void initTexCoord(){/*...*/}
    public void convertTo(ref Mesh mesh){/*...*/}
    public void updateTexCoordTo(ref Mesh mesh){/*...*/}
}
```

用的是老师提供的结构，并且补充了`TexCoord`，用来放参数化坐标。

3.2.2 MeshAPI_CPP

```
public unsafe struct MeshAPI_CPP
{
    public int VerticesNumber;
    public int FaceNumber;
    public int TexCoordNumber;

    public float* VerticesX;
    public float* VerticesY;
    public float* VerticesZ;

    public int* FaceA;
    public int* FaceB;
    public int* FaceC;

    public float* TexCoordX;
    public float* TexCoordY;

    public void linkWith(ref MeshAPI_CS api_cs){/*...*/}
}
```

作用类似`WarpAPI_CPP`。

3.2.3 MeshAPI

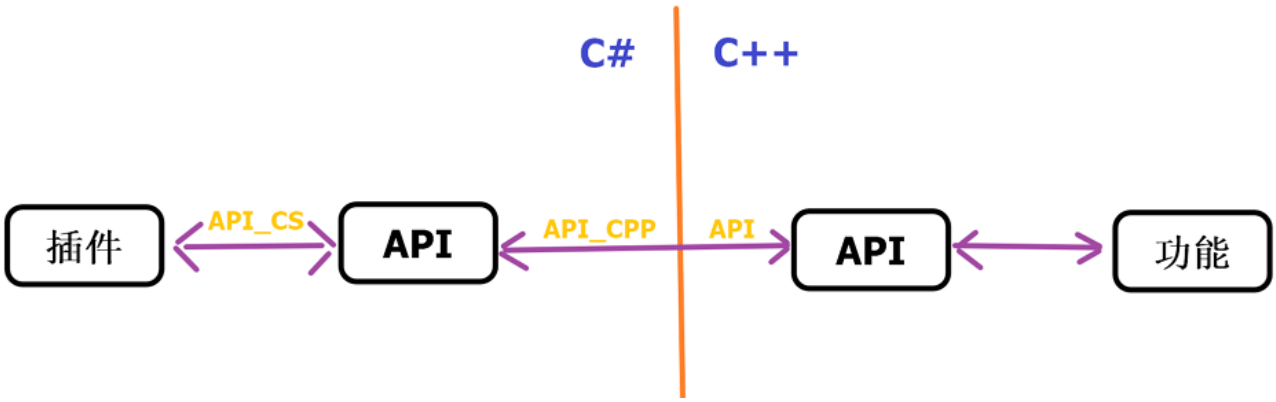
```
struct MeshAPI
{
    int VerticesNumber;
    int FaceNumber;
    int TexCoordNumber;

    float* VerticesX;
    float* VerticesY;
    float* VerticesZ;

    int* FaceA;
    int* FaceB;
    int* FaceC;

    float* TexCoordX;
    float* TexCoordY;
    // method
    void convertTo(Mesh3D & mesh);
    void updateVFrom(Mesh3D & mesh);
    void updateTFrom(Mesh3D & mesh);
    /*-- only load V and F --*/
    void loadFrom(Mesh3D & mesh);
};
```

作用类似`WarpAPI`。

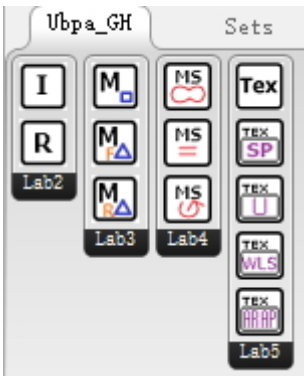
4. 调用关系

插件代码跟`C#`的`API_CS`交互。`C#`的`API`就管理`API_CS`和`API_CPP`，`API_CPP`用来跟`C++`的`API`对接。`C++`里实现了调用作业2-5相关功能的函数，供`C#`调用。

以上的表述可以用下边的图来表示



5. 插件面板



归属在了组`Ubpa_GH`中，子组分别为`Lab2`、`Lab3`、`Lab4`和`Lab5`。

按子组从上到下说明一下功能

- Lab2：IDW、RBF
- Lab3：矩行网格、随机三角网格、特征点三角网格
- Lab4：闭合曲线最小曲面、全局法、局部法
- Lab5：简单贴图、SP、Uniform、WLS、ARAP

其中"**简单贴图**"是跟以往作业无关的一个功能。作业5实现了4种参数化方法，并最后将得到参数化坐标写入了`mesh`的`texturecoordinates`中。原来的实验中OpenGL实现了利用参数化坐标进行了贴图，而这次作业没法使用OpenGL了。因此我只是简单的利用`mesh`的`colors`来进行简单的**点颜色**赋值。`grassshopper`会根据点颜色来插值网格内的颜色，只不过这个方法很**粗糙**。

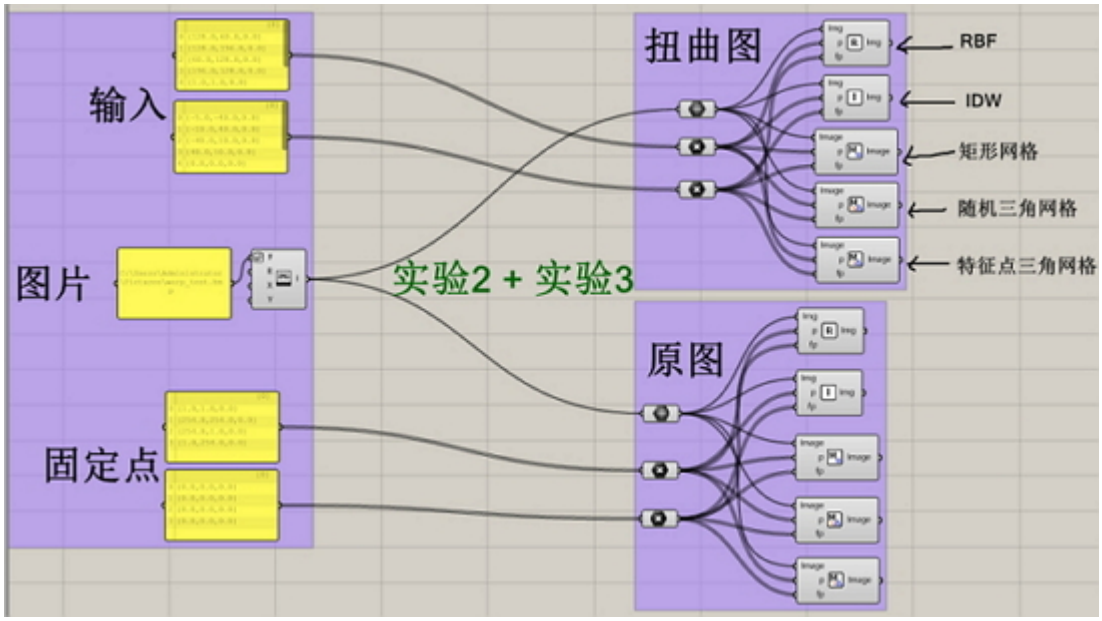
6. 功能测试

主要是为了验证**正确性**，所达成的效果可能没有原作业的好。

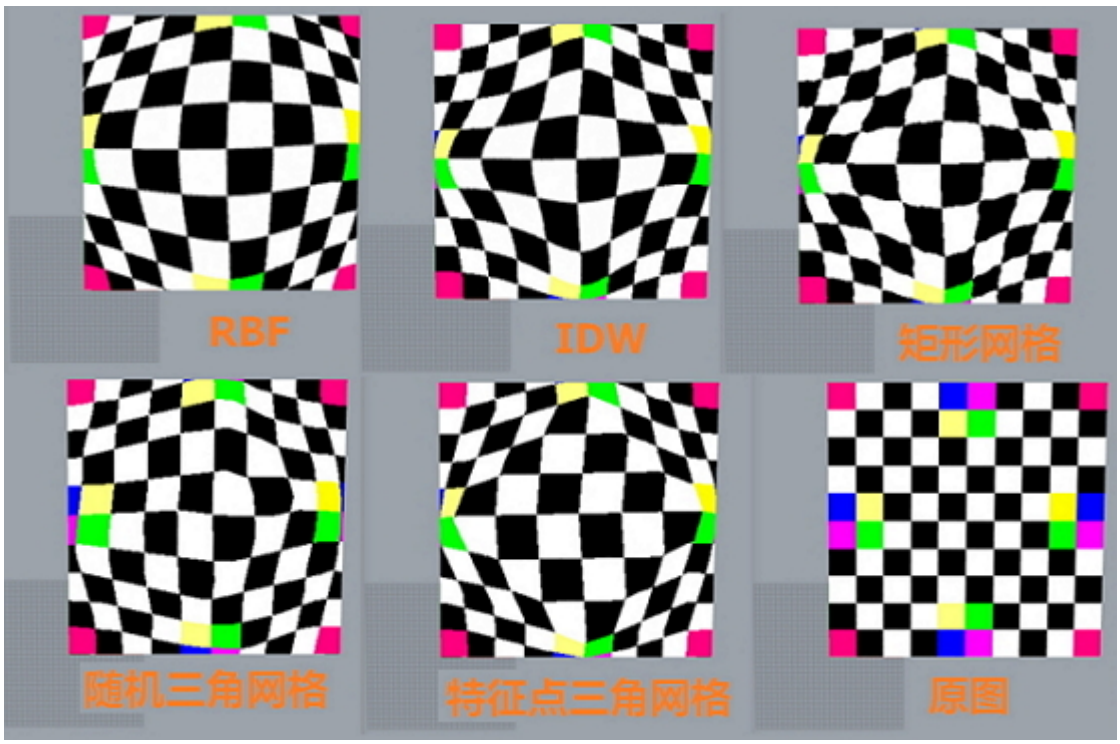
6.1 作业2 和 作业3

扭曲图像：IDW、RBF、矩形网格、随机三角网格、特征点三角网格

6.1.1 例子



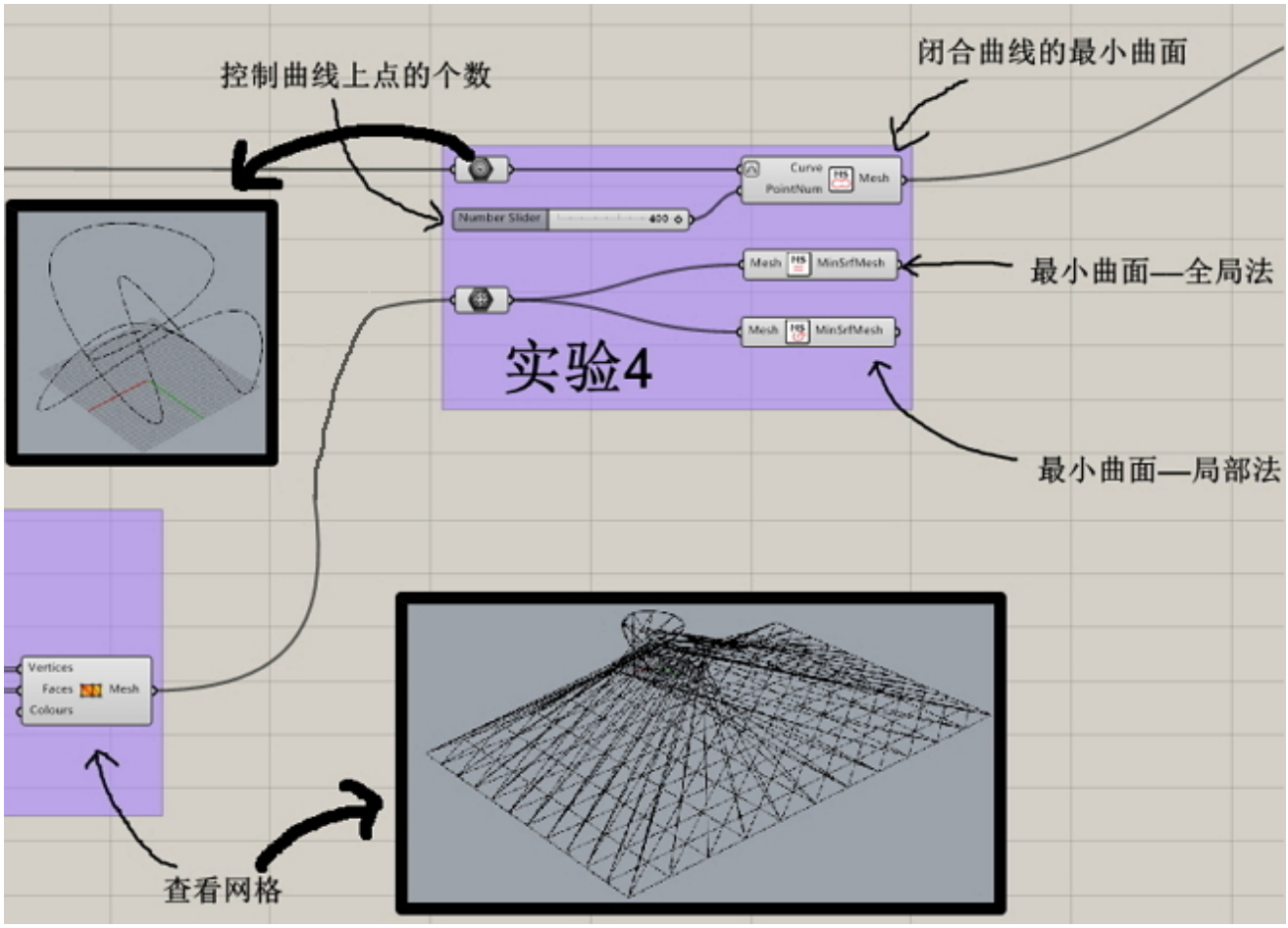
6.1.2 结果



6.2 作业4

最小曲面：局部法、全局法、闭合曲线最小曲面

6.2.1 测试例子



图中的网格就是左上角**闭合曲线相应的网格**

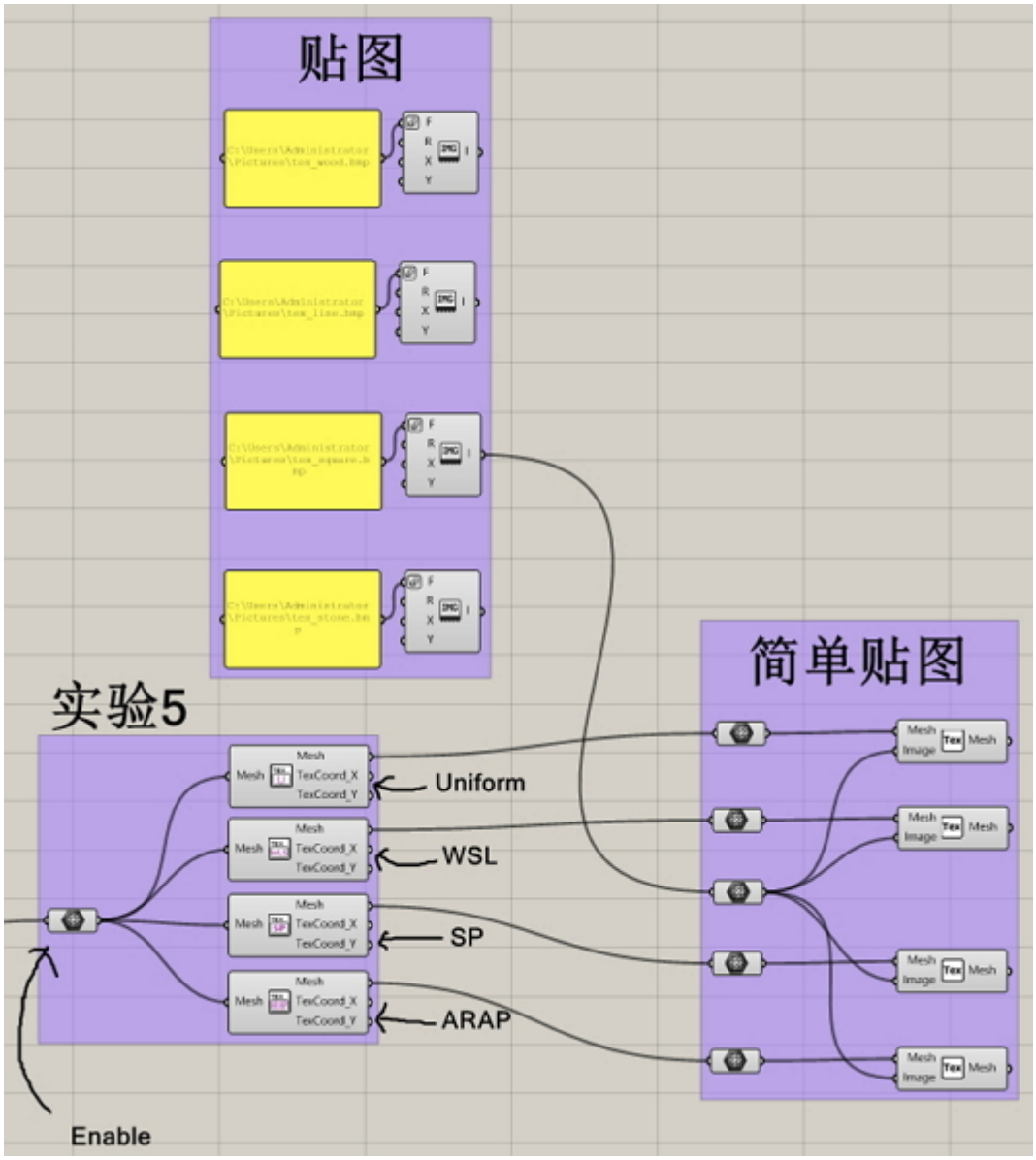
6.2.2 结果

6.3 作业5

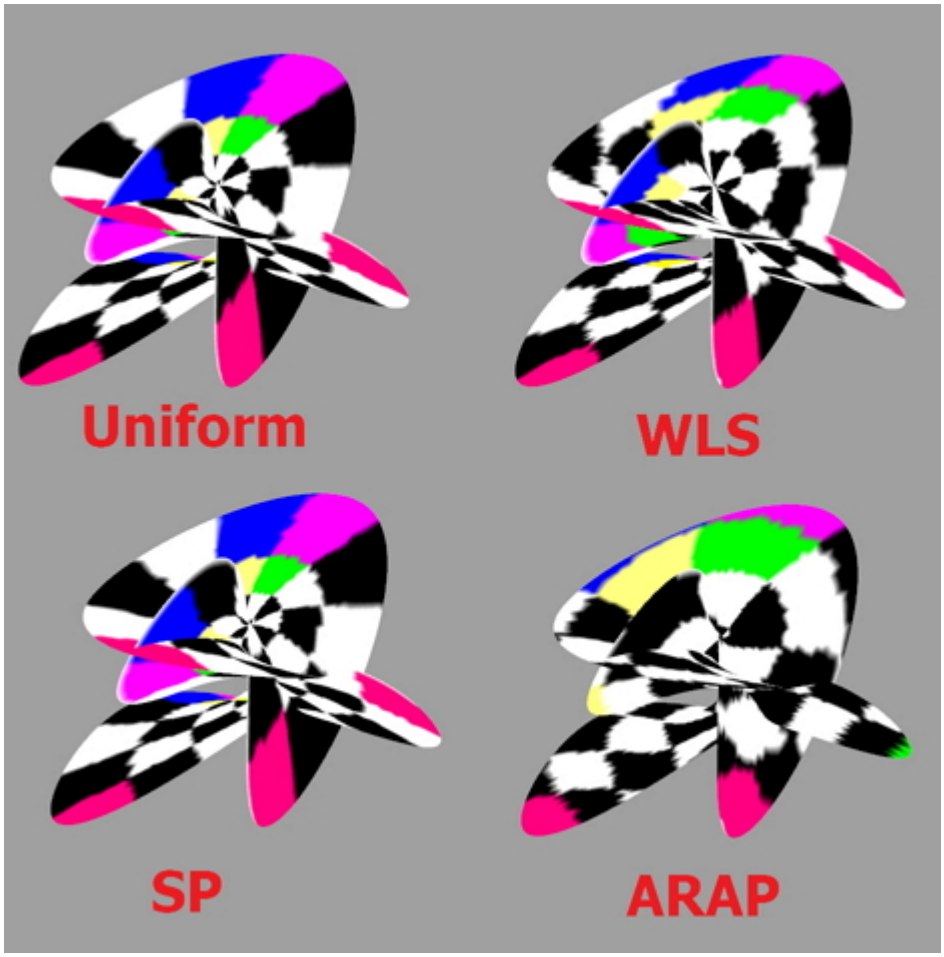
参数化：Uniform、WSL、SP、ARAP

结果存放在了`mesh`的`texturecoordinates`里，使用**简单贴图**查看结果。

6.3.1 例子



6.3.2 结果

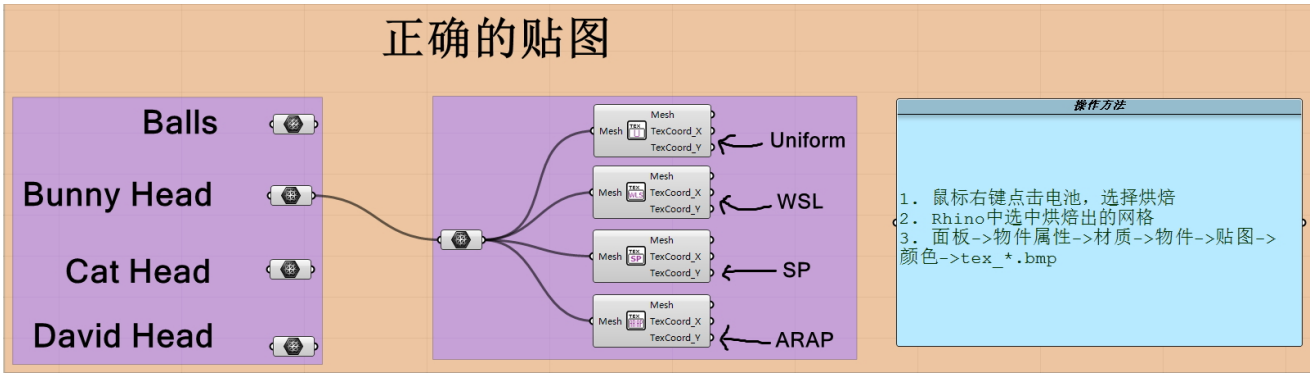


6.3.3 贴图改进

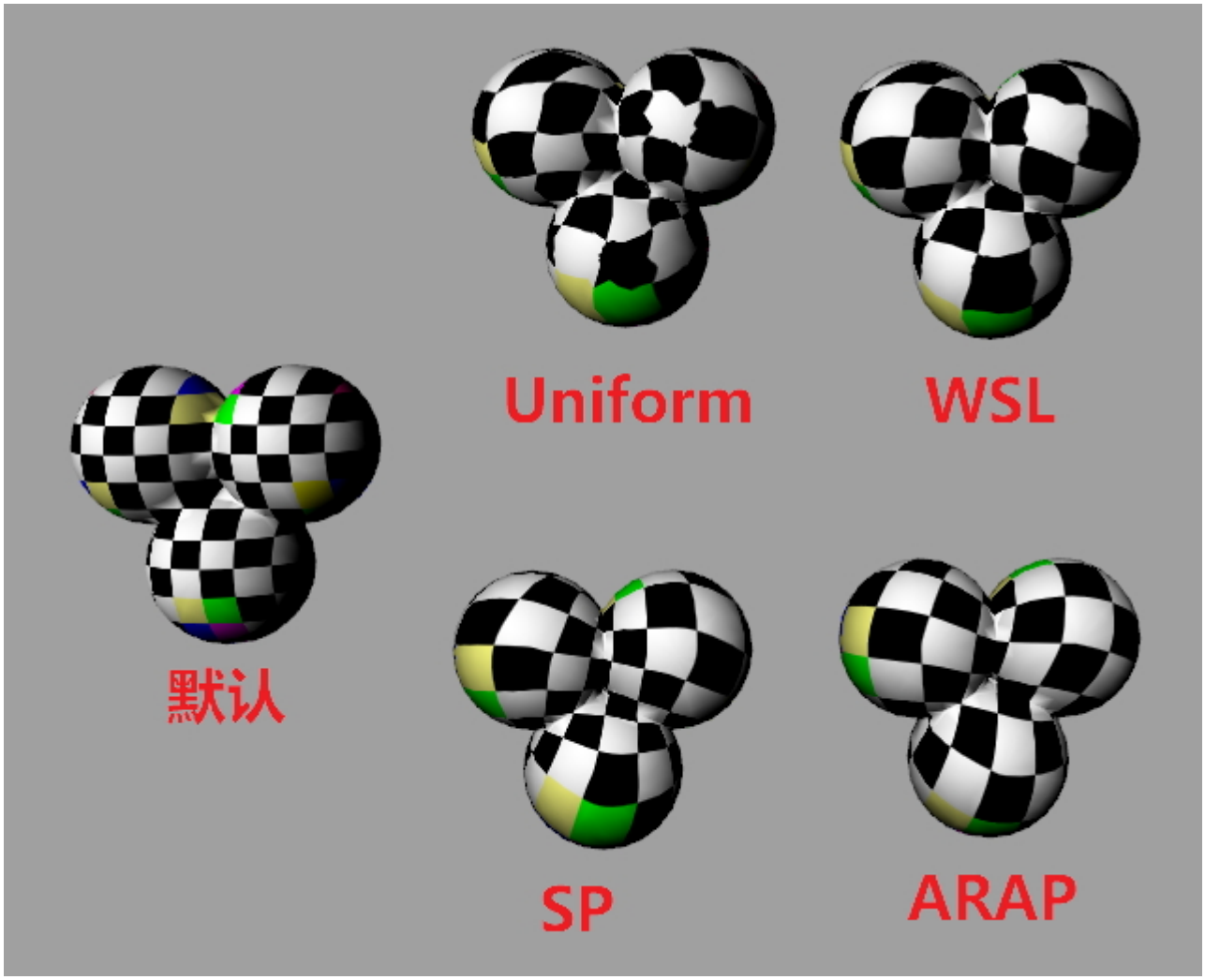
> 2018/04/22

之前说到用点颜色来进行贴图，后来经过探索后终于知道了关于贴图的正确方法。

6.3.3.1 例子



6.3.3.2 结果



7. 细节问题

7.1 内存问题

`C#`有垃圾收集机制，因此我尽量都是在`C#`里分配内存，在`C++`里直接使用空间就行了。

对于作业2和作业3，`C++`函数做的是将图像的颜色进行修改，因此可以在原空间处理，不需要分配额外内存。

对于作业4的全局法和局部法，`C++`函数做的是将`Mesh`各点的坐标进行修改。

对于作业5，`C++`函数做的就是获取参数化坐标，数量就是**点的数量**，因此可以在`C#`里分配空间，在`C++`里直接使用。

唯一一个做不到`C#`里分配空间的是作业4的一个功能——**求闭合曲线的最小曲面**。因为无法提前知道生成的所生成的曲面的点的数量（依赖于底层的具体实现）。因此需要`C++`里分配空间。因此为了释放空间，只能提供一个`C++`函数在`C#`里主动释放。

7.2 OpenCV兼容性问题

经过测试，发现只要代码中有任何调用了opencv的代码，插件在载入dll时会报**格式不正确**的错误。好在程序中只使用了OpenCV的求特征点的方法。解决办法就是用个人实现的C++特征点检测算法FAST即可。

7.3 Qt的脱离

作业2和作业3的`Warp`类使用了`QT`的`QImage`。为了代码能够不依赖`Qt`，自己实现了简单的`Img`类，实现了相关的函数。

8. 使用方法

8.1 编译

> 可以跳过这一步骤，因为压缩包里已经含有`CPP.dll`和`Ubpa_GH.gha`了

解决方案目录是`Ubpa_GH`，因为整个解决方案里边没有用`Qt`和`OpenCV`，只是用了`eigen`，并且我已经把`eigen`放到了目录里边，因此一般来说是可以直接使用`vs2010`编译的。

编译后

- `C++`的`CPP.dll`文件在`Ubpa_GH\x64\Realease`中
- `C#`的`Ubpa_GH.gha`文件在`Ubpa_GH\ Ubpa_GH\bin\x64\Realease`中

8.2 文件放置

- 将`ubpa.gha`放在`C:\Users\Username\AppData\Roaming\Grasshopper\Libraries`中
- 将第三方插件`MeshTools.gha`和`MeshEdit.gha`也放在`C:\Users\Username\AppData\Roaming\Grasshopper\Libraries`中
- 将`DLL`所在目录加入到系统环境变量`Path`中。

> 我的电脑-->属性-->高级系统设置-->环境变量-->系统变量-->Path

- 打开`Grasshoper`，然后再打开`ubpa.gh`，就能看到各插件的使用例子了。报告中使用的例子就来自该文件。插件在`Ubpa_GH`组里。