

作业要求及框架说明

目录

作业要求及框架说明.....	1
现有框架.....	1
作业要求.....	1
作业框架补充使用说明.....	2
Ubuntu 运行环境:	2
Windows 运行环境:	3
接口说明:.....	5
数据结构样例:	5
NURBS 曲线曲面上的点的计算验证方法:.....	8
NURBS 曲线曲面上某点导数的计算验证方法:.....	8
NURBS 曲面上某点法向的计算验证方法:.....	8

现有框架

1、一个可视化的界面

- 一个图形界面，包括显示、光照等功能；鼠标点击旋转、鼠标滚轮放缩、键盘 wasd 键平移事件。
- 一个菜单栏，包括选择是否显示光照曲面、曲线曲面控制网格、曲面的参数曲线、曲线曲面导数、曲面法向等功能。

2、给出基本数据结构 如：3D NURBS 曲线、3D NURBS 曲面，下文给出样例。

3、开发平台

Window: cmake 建议用 Visual Studio 自带的

Ubuntu: cmake

作业要求

注意，作业必须要在给出的框架内完成。

首先，根据 Sub-routines/functions requirements 要求，编写函数。其中，框架中已经包含部分功能，这些功能需要自己重新编写函数实现框架中的以下功能，并替换掉：

NURBS 曲线、曲面求值 `tinynurbs::curvePoint`、`tinynurbs::surfacePoint`

NURBS 曲线、曲面求导数 `tinynurbs::curveDerivatives`、`tinynurbs::surfaceDerivatives`

NURBS 曲面求法向 `tinynurbs::surfaceNormal`

注意，实现的过程中不能调用 `tinynurbs::basic.h` 中的方法，否则按抄袭处理！

然后，实现以下三组插值点序列的 B-样条曲线插值，并计算 4 种参数化方法所得相应插值曲线的延展能量和弯曲能量。

插值点序列 1：

$Q_1 = \{\{0, 0, 5\}, \{0, 3, 5\}, \{0, 4, 5\}, \{0, 5, 5\}, \{0, 7, 5\}, \{0, 11, 5\}, \{0, 13, 5\}, \{0, 15, 5\}, \{5, 15, 5\}, \{5, 13, 4\}\}$

插值点序列 2:

$Q_2 = \{\{0, 0, 0\}, \{2, -2, -1\}, \{4, -4, -2\}, \{5, -2, -3\}, \{7, -1, -4\}, \{8, 0, -5\}, \{10, 1, -6\}, \{11, 0, -7\}, \{11, 4, -8\}, \{12, 4, -9\}, \{12, 9, -10\}, \{13, 8, -12\}, \{12, 14, -8\}, \{13, 12, -10\}\}$

插值点序列 3:

$Q_3 = \{\{2, 3, 4\}, \{0, 3, 4\}, \{0, 3, 0\}, \{3, 3, 0\}, \{3, 0, 0\}, \{3, 0, 2\}, \{5, 0, 2\}, \{4, -2, 0\}, \{5, 2, 0\}, \{6, -1, 0\}, \{7, 0, 0\}\}$

注意，以上自己编写的函数可以复用在这里。

作业框架补充使用说明

Ubuntu 运行环境:

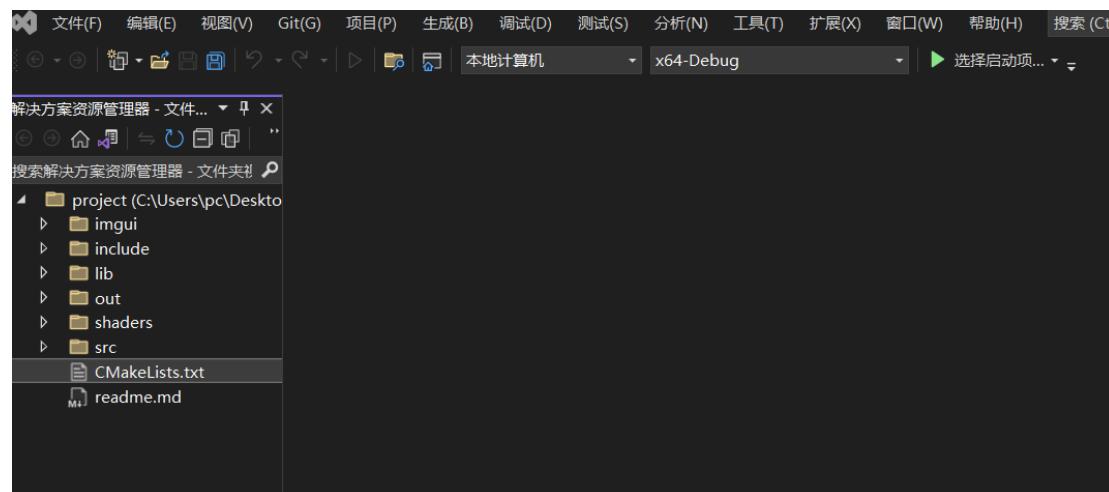
参考文件夹下的 readme.md

Windows 运行环境:

Visual Studio2022 直接用打开本地文件夹打开 project 文件夹



加载完毕后点击 CmakeList.txt:



可能会提示你生成要刷新的 CMake 缓存，点击生成如下图，也可能它会自动生成，如果自动生成只需要等待一下，命令行出现 CMake 生成完毕提示就可以如下图：

```

1 cmake_minimum_required (VERSION 3.16)
2 project ("project")
3
4 file(GLOB SRCS "./src/*.cpp" "./src/*.c" "./imgui/*.cpp" "./imgui/backends/*.cpp")
5 add_executable(project ${SRCS})
6 target_include_directories(project PRIVATE "./include" "./imgui" "./imgui/backends")
7 if(LINUX)
8   target_link_libraries(project dl GL glfw)
9 else()
10  target_link_directories(project PRIVATE "./lib")
11  target_link_libraries(project glfw3)
12 endif()
13

```

输出

显示输出来源(S): CMake

```

1> [CMake] -- Configuring done
1> [CMake] -- Generating done
1> [CMake] -- Build files have been written to: C:/Users/pc/Desktop/project(1)/project/out/build/x64-Debug
1> 已提取 CMake 变量。
1> 已提取源文件和标头。
1> 已提取代码模型。
1> 已提取工具链配置。
1> 已提取包含路径。
1> CMake 生成完毕。

```

然后在选择启动项那里选择 project.exe，如下图：

项目

显示/隐藏调试目标...

当前文档(CMakeLists.txt) -> Debug

显示当前文档

无最近启动项目

project

project

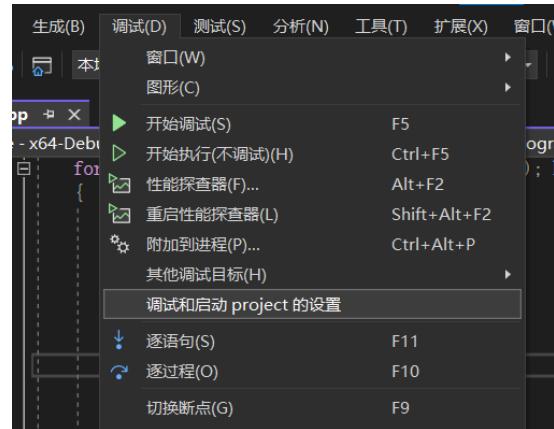
然后点击运行，首次运行可能需要等待一会，可能会有目录出错的情况如下图：

Microsoft Visual Studio 调试控制台

ERROR: SHADER FILE WAS UNSUCCESSFULLY READ

C:\Users\pc\Desktop\project(1)\project\out\build\x64-Debug\project.exe (进程 25912)已退出，代码为 -1。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口...

错误修正设置方式如下：



如上图，先打开调试和启动 project 的设置，会出现 launch.vs.json 文件
然后在文件里面加入一行: "currentDir": "\${projectDir}/src" 即可
修改后如下图所示：

```
launch.vs.json ✘ ×
架构: ..\..\..\AppData\Local\Microsoft\VisualStudio\17.0_aab75ff3\OpenFolder\launch_schema.json
1  {
2   "version": "0.2.1",
3   "defaults": {},
4   "configurations": [
5     {
6       "type": "default",
7       "project": "CMakeLists.txt",
8       "projectTarget": "project.exe",
9       "name": "project.exe",
10      "currentDir": "${projectDir}/src"
11    }
12  ]
13 ⚡ }
```

接口说明：

main.cpp 函数里 GLprogram 类的 initial 函数和 run 函数里传入两个参数

- Vector<tinyNurbs::RationalCurve> : 需要绘制的有理 b 样条曲线
- Vector <tinyNurbs::RationalSurface>： 需要绘制的有理 b 样条曲面

数据结构样例：

NURBS 曲线：

```
tinynurbs::RationalCurve<double> crv; // Planar curve using float32
crv.control_points = { glm::vec3(0, 0, 5), // std::vector of 3D points
    glm::vec3(0, 3, 4),
    glm::vec3(0, 4, 3),
    glm::vec3(0, 5, 6),
    glm::vec3(0, 7, 5),
    glm::vec3(0, 11, 3),
    glm::vec3(0, 13, 4),
    glm::vec3(0, 15, 5),
};

crv.knots = { 0, 0, 0, 0, 1.0 / 5, 2.0 / 5, 3.0 / 5, 4.0 / 5, 1, 1, 1, 1 }; // std::vector of floats
crv.degree = 3;
crv.weights = { 0.2, 0.3, 0.4, 1, 0.5, 0.6, 0.7, 0.8 };
```



NURBS 曲面:

```

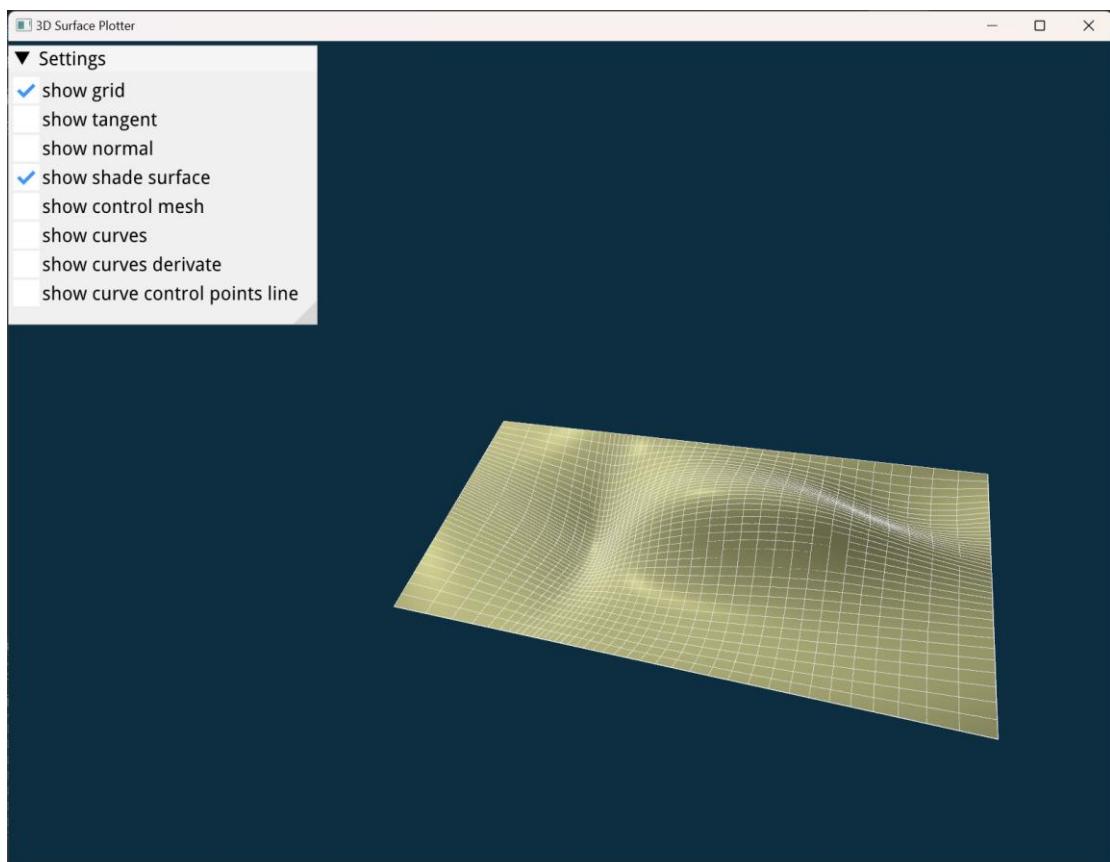
tinynurbs::RationalSurface<double> srf3;

srf3.degree_u = 3;
srf3.degree_v = 3;
srf3.knots_u = { 0, 0, 0, 0, 0.2, 0.4, 0.6, 0.8, 1, 1, 1, 1 };
srf3.knots_v = { 0, 0, 0, 0, 0.5, 1, 1, 1, 1 };

srf3.control_points =
{ 8, 5,
{
    glm::fvec3(0, 0, 5), glm::fvec3(2, 0, 5), glm::fvec3(3, 0, 5), glm::fvec3(5, 0, 5), glm::fvec3(8, 0, 5),
    glm::fvec3(0, 3, 5), glm::fvec3(2, 3, 4), glm::fvec3(3, 3, 3), glm::fvec3(5, 3, 4), glm::fvec3(8, 3, 5),
    glm::fvec3(0, 4, 5), glm::fvec3(2, 4, 3), glm::fvec3(3, 4, 1), glm::fvec3(5, 4, 3), glm::fvec3(8, 4, 5),
    glm::fvec3(0, 5, 5), glm::fvec3(2, 5, 4), glm::fvec3(3, 5, 4), glm::fvec3(5, 5, 3), glm::fvec3(8, 5, 5),
    glm::fvec3(0, 7, 5), glm::fvec3(2, 7, 6), glm::fvec3(3, 7, 7), glm::fvec3(5, 7, 2), glm::fvec3(8, 7, 5),
    glm::fvec3(0, 11, 5), glm::fvec3(2, 11, 5), glm::fvec3(3, 11, 9), glm::fvec3(5, 11, 3), glm::fvec3(8, 11, 5),
    glm::fvec3(0, 13, 5), glm::fvec3(2, 13, 3), glm::fvec3(3, 13, 7), glm::fvec3(5, 13, 4), glm::fvec3(8, 13, 5),
    glm::fvec3(0, 15, 5), glm::fvec3(2, 15, 5), glm::fvec3(3, 15, 5), glm::fvec3(5, 15, 5), glm::fvec3(8, 15, 5)
}
};

srf3.weights =
{ 8, 5,
{
    1, 1, 1, 1, 1,
    1, 1, 1, 1, 1,
    1, 1, 1, 1, 1,
    1, 1, 1, 1, 1,
    1, 1, 1, 1, 1,
    1, 1, 1, 1, 1,
    1, 1, 1, 1, 1,
    1, 1, 1, 1, 1
}
};

```



除此之外，**框架中还有其他几个样例**，其中 crv1 - crv5 为任务 1 中的五条 NURBS 曲线

NURBS 曲线曲面上的点的计算验证方法：

在 GLprogram.cpp 文件里面用自己写的 NURBS 曲线曲面上的点的计算函数替代

`tinynurbs::curvePoint` 函数

`tinynurbs::surfacePoint` 函数

数据结构可沿用 `tinynurbs` 中 NURBS 曲线曲面定义，但返回值必须是 `glm::vec3` 类型表示三维点的坐标

NURBS 曲线曲面上某点导数的计算验证方法：

在 GLprogram.cpp 文件里面用自己写的 NURBS 曲线曲面上的点的导数的计算函数替代

`tinynurbs::surfaceDerivatives` 函数

`tinynurbs::curveDerivatives` 函数

数据结构可沿用 `tinynurbs` 中 NURBS 曲线曲面定义

返回值可自己设计，曲线要保证能把导数结果转化为 `glm::vec3` 类型，表示三维向量；曲面因为是 u v 两个方向的导数，所以是要能分别转化为 u v 两方向的 `glm::vec3` 类型。

NURBS 曲面上某点法向的计算验证方法：

在 GLprogram.cpp 文件里面，用自己写的 NURBS 曲线曲面上的点的法向的计算函数替代

`tinynurbs::surfaceNormal` 函数

数据结构可沿用 `tinynurbs` 中 NURBS 曲线曲面定义

返回值需要是 `glm::vec3` 类型的法向方向