图形学实验 PA4：自由体变形

指导教师：胡事民　助教：黄家晖

2019 年 11 月 18 日

1　实验综述

本次作业中，你将在之前完成的代码框架中再加入一项新的功能：允许用户交互的自由体变形（FreeForm Deformation）算法，这种算法是许多三维网格变形动画的基础。通过拖动张量积 Bezier 体上的控制点，其内嵌网格的形状也会发生相应的变化。

2　细节说明

自由体变形算法（FreeForm Deformation，后简称 FFD）最早由美国杨百翰大学（BYU-Provo）的 Thomas W. Sederberg 教授于 1986 年提出，论文名称为 Free-Form Deformation of Solid Geometric Models。这种算法思想简单，易于实现，目前是许多三维动画软件中的基本变形算法（例如 Blender 中的 Cage 修改器）。

FFD 的基本思想是将待变形的几何形状 $\mathcal{M}$ “嵌入”一个由三维控制点阵列组成的网格 $\mathcal{T}$（张量积）中，用户通过拖动 $\mathcal{T}$ 上的控制点，控制 $\mathcal{M}$ 的形状，其效果如图1所示。在算法初始化的时候，FFD 首先为 $\mathcal{M}$ 中的每一个顶点 $\boldsymbol{v}_i = (x_i, y_i, z_i) \in \mathbb{R}^3$ 计算局部参数 $\boldsymbol{t}_i = (u_i, v_i, w_i) \in [0,1]^3$，满足 $\forall i, f_{\mathcal{C}}(\boldsymbol{t}_i) = \boldsymbol{v}_i$，其中 $f_{\mathcal{C}}$ 是基于初始控制点集合 $\mathcal{C}$ 的一个张量积映射函数。当用户操纵了控制点之后，原始 $\mathcal{C}$ 变为了 $\mathcal{C}'$，那么 $\mathcal{M}$ 中每一个顶点新的位置则会使用 $\boldsymbol{v}'_i = f_{\mathcal{C}'}(\boldsymbol{t}_i)$ 来计算。具体 f 的定义取决于所使用的张量积类型，在本次 PA 中我们使用课上所讲的张量积 *Bezier* 体。

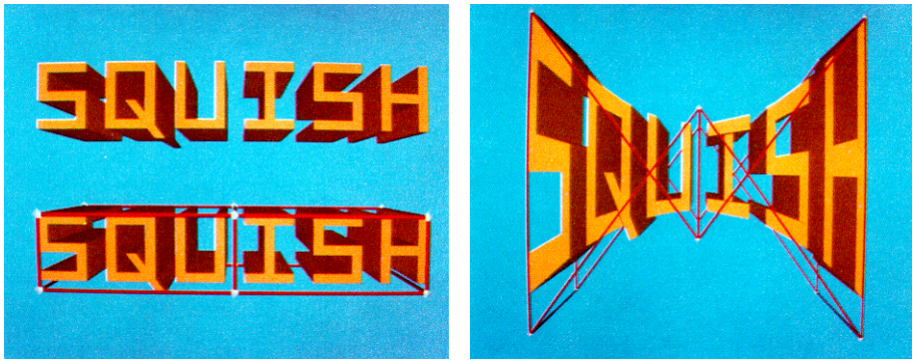

图 1: FFD 算法示例。左图：原始形状以及控制点网格；右图：变形后的形状和控制点网格。

设张量积 Bezier 体所占据的空间为 $[X^-, X^+] \times [Y^-, Y^+] \times [Z^-, Z^+]$（一般设置为 $\mathcal{M}$ 的包围盒），其对应控制点集合 $\mathcal{C}$ 的三维阵列控制点个数为 $(L+1) \times (M+1) \times (N+1)$，每个控制点的坐标为 $\boldsymbol{P}_{lmn} (l = 0, \ldots, L; m = 0, \ldots, M; n = 0, \ldots, N)$。则使用 FFD 算法的求解过程描述如下：

- **初始化**：首先对 $\mathcal{M}$ 上的每一个顶点 $\boldsymbol{x}_i$ 计算局部参数 $u_i = \frac{x_i - X^-}{X^+ - X^-}, v_i = \frac{y_i - Y^-}{Y^+ - Y^-}, w_i = \frac{z_i - Z^-}{Z^+ - Z^-}$。
- **更新**：当用户对控制点的坐标进行了操作之后 $\{\boldsymbol{P}_{lmn}\} \to \{\boldsymbol{P}'_{lmn}\}$，新的 $\mathcal{M}'$ 中每个顶点 $\boldsymbol{v}'_i$ 的计算公式为：

$$\boldsymbol{v}'_i = f_{\mathcal{C}'}(u_i, v_i, w_i) = \sum_{l=0}^{L} \sum_{m=0}^{M} \sum_{n=0}^{N} B_{l,L}(u_i) B_{m,M}(v_i) B_{n,N}(w_i) \boldsymbol{P}'_{lmn}, \tag{1}$$

其中 $B_{l,N}(\cdot)$ 是上一次 PA3 中介绍过的 Bezier 曲线基函数（参考 PA3 中的公式 (2)）。整体的计算可以使用三重循环实现。

3 框架代码说明

3.1 环境配置与编译

我们推荐使用带有 `CMake` 套件的 `Ubuntu` 系统进行编程，`Windows 10` 下可以使用 Ubuntu Subsystem。我们的框架代码依赖于 `GL` 和 `GLUT`，前者一般在安装显卡驱动的时候会自动配置好，如果 `CMake` 配置出错可以尝试使用 `apt` 安装 `mesa-common-dev`；后者请使用 `apt` 安装 `freeglut3-dev`。安装完成之后，请在包含有 `run_all.sh` 的文件夹下打开终端，并执行：

```
bash ./run_all.sh
```

这段脚本会自动设置编译，并在提供的 4 个测例上运行你的程序。你的程序最终会被编译到 `bin/`文件夹中，而输出的渲染图片位置在 `output/`文件夹中。

3.2 交互模式操作说明

框架代码的命令行参数格式和 PA2 相同，第一个参数为样例 txt 文件的路径，第二个参数是输出图片文件的路径，如果第二个参数留空则会进入交互模式。交互模式被分为了两种状态：观察状态（画面背景色为场景设定好的颜色）和编辑状态（画面背景色为黑色），二者使用键盘 Z 键进行切换。

观察状态使用鼠标操控，各个按键的作用请参见 PA2 中的描述。编辑状态下用户能够使用鼠标拖动 FFD 自由变形体的控制点 $\boldsymbol{P}_{lmn}$ 的位置，如果你的代码实现正确，则也能够观察到网格随着鼠标拖动发生相应的变形，如图2所示。

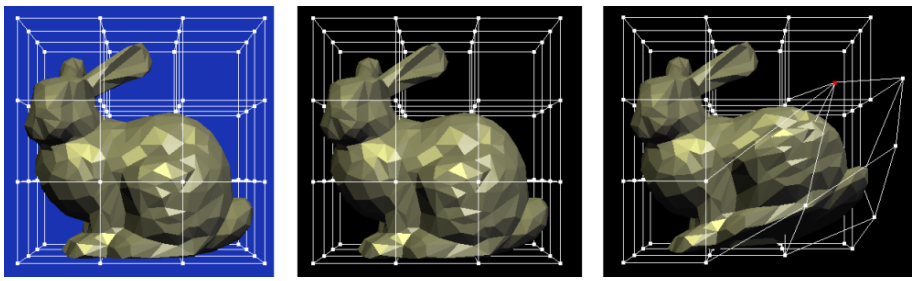

图 2: 交互模式操作示例。切换到编辑状态之后背景变为黑色，之后就可以使用鼠标直接拖动控制点的位置了。再次按 Z 键可以切换回观察状态。

3.3 实现步骤

首先，你需要像以往一样使用 meld 程序将 PA3 的代码合并过来，确保前两个测例能够输出正确的结果。

本次作业所有需要新填写的地方都被标记了 `TODO (PA4)`，请全文查找该字样，每实现完成一个功能，你就可以去掉一个 TODO。你需要实现的主要的函数是 `ffd_cage.hpp` 中的 `initialize` 和 `update` 函数，这两个函数分别代表2节中的**初始化**步骤和**更新**步骤，你需要了解各个变量的含义，以及他们怎样和具体的数学公式对应。由于涉及到 Bezier 基函数的计算，你可以将上一次 PA 的相关代码复用进来。

4 测试用例

为了测试代码是否正确无误，我们总共提供了 4 个测试用例，第一个测例和之前的相同，第二个测例（图3）本身不包含任何变形，供给同学们自由测试时使用，后 2 个（如图4,5所示）是本次作业新添加的测例，包含了预定义好的控制点变形输入。你也可以根据场景的文件格式构造样例进行自我测试。

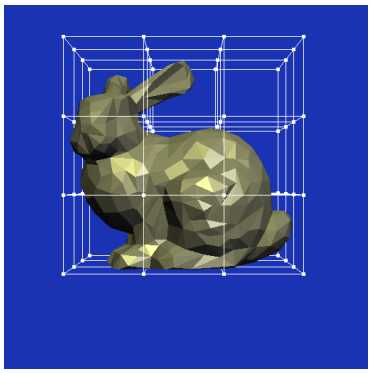

图 3: 网格中的兔子

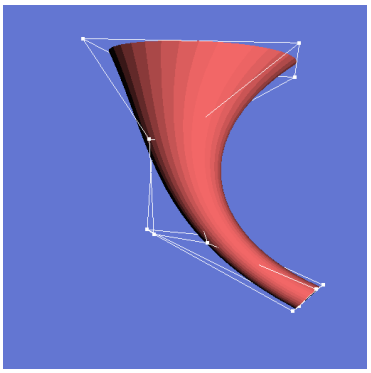

图 4: 迷幻烟斗

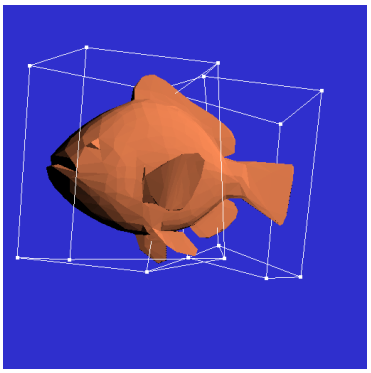

图 5: 游泳的鱼

5 提交说明

请将你的代码放入 `code` 文件夹（无须包含可执行文件和输出结果），和一个 `REPORT.pdf` 文档一起打包成 `zip` 文件提交至网络学堂。具体文件树结构如下所示：[1]

[1] 由于助教会使用自动化测试脚本来运行大家的程序，因此请务必遵循该文件树。文件夹包含 MacOS 生成的 `.DS_STORE` 文件不影响评分。

```
姓名-学号-PA4.zip
├─ REPORT.pdf
├─ code
│  ├─ run_all.sh
│  ├─ CMakeLists.txt
│  ├─ src/
│  ├─ include/
│  └─ ...
└─ ... (其他你想放的文件)
```

在 `REPORT.pdf` 文档中主要回答以下问题：

- 本次实现的 FFD 算法在实际应用中有哪些问题？可以怎样改进？
- 你在完成作业的时候和哪些同学进行了怎样的讨论？是否借鉴了网上/别的同学的代码？
- (可选) 你对本次编程作业有什么建议？文档或代码中有哪些需要我们改进的地方？

本次作业的 Deadline 以网络学堂为准。迟交的同学将得到一定的惩罚：晚交 3 天内分数将降低为 80%，3 天以上 1 周以内分数降为 50%，迟交一周以上的同学分数为 0。

在检查你的作业时，助教的自动化脚本会在 `code` 目录下运行 `run_all.sh` 文件，不能成功编译者可能会被酌情扣分。最后，欢迎同学们在压缩包中加入自己设计的场景 `txt` 文件，你的场景可能会被用于下学期的课堂作业教学中哦。