2020-2021-1 SE344 计算机图形学 期末大作业报告

——刘千禧（517021910785）

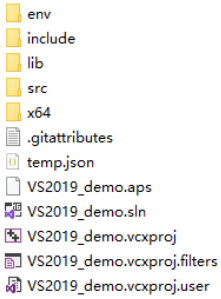
目录

一、综述

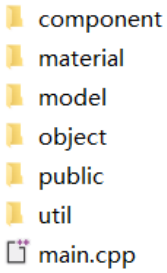
1. 本报告是 2020-2021 学年第一学期，计算机图形学课程（SE344）期末大作业报告。报告共分为三个部分，第一部分说明整个程序的结构设计，第二部分按照作业评分标准详细说明各个功能具体的实现方式，第三个部分说明其他程序相关的内容（包括操作说明和编程环境说明）。

二、程序结构设计

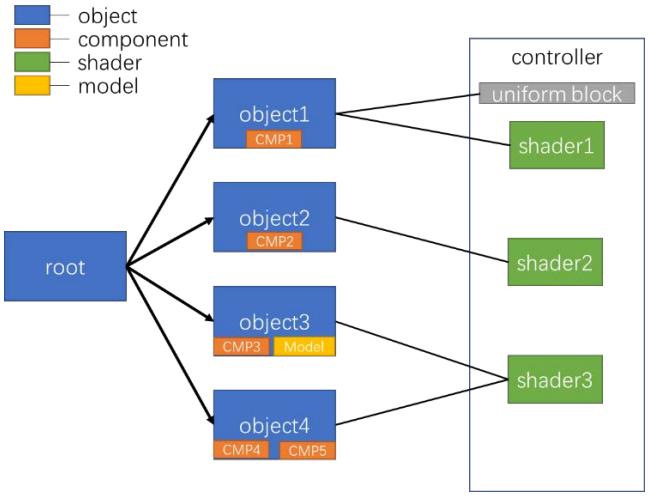
1. 项目目录结构说明
   1) 代码目录图（./code 目录下）



   2) 各个文件（夹）存放的内容如下
      a) ./env：运行环境，里面存放了编译完成的 X64-Release 版本可执行文件（包括提交版和演示版，两者的区别将在 3.2.2 小节中详细介绍）和运行需要的文件（X64-Release 版本的动态链接库，模型文件，材质文件，着色器程序源文件）；
      b) ./include：存放了 glfw、glad、assimp、glm 等工具库的头文件；
      c) ./lib：存放静态链接库；
      d) ./src：存放源代码；
      e) 其他：VS2019 工程文件，git 相关文件；
2. 模块化程序结构说明
   1) 简述
      a) 为了能够更好地使用 OpenGL 并复用已有代码，我模仿 Unity 的风格将整个工程进行模块化搭建。
      b) 文件目录（./code/src 目录下）



      c) 一共有组件（component）、材质（metarial）、模型（model）、物体（object）四个模块，各个模块的源文件位于文件夹中，除此之外./public 文件夹中存放了一些渲染引擎封装文件和场景生成的代码，./util 文件夹中存放了一些自定义的工具。
   2) 各个模块之间的组织结构
      a) 组织结构如图所示



3) 物体（object）：实现物体抽象和组织
   a) 使用树状结构组织物体，物体的各个操作将从根物体开始逐个调用至叶子节点；
   b) 每一个物体都可以拥有若干组件实现相应的功能；
   c) 每一个物体都拥有自身的着色器，多个物体可以共用同一个着色器
   d) 物体可以持有实例化的模型模块用于模型加载；
   e) 物体的 start()初始化物体，可能向着色器传输顶点数据和 uniform 数据或控制组件向着色器传送顶点数据和 uniform 数据；
   f) 物体的 update()先调用自身所有组件的更新，再调用子物体的更新；
   g) 物体的 render()向着色器传送顶点数据和 uniform 数据或控制组件向着色器传送顶点数据和 uniform 数据；
   h) 在渲染循环前，物体的 start()将被调用，在渲染循环中，物体先完成 update()，再完成 render()；
   i) 与 Unity 相似，物体树上的子物体的 Transform 组件会受到父物体 Transform 组件的影响（Transform 组件控制了物体的位置、旋转欧拉角和缩放比例）
4) 组件（component）：实现物品的各个功能
   a) 每个组件将被挂载在物体上，实现相应的功能，一个物体不能挂载多个同类型组件，但是可以挂载多个不同类型的组件；
   b) 组件的 start()将在渲染循环前调用，实现组件初始化；
   c) 组件的 update()函数将每帧被调用，实现组件数据更新；
   d) 组件的 render()函数将在每帧被调用，将顶点数据和 uniform 数据传输至着色器，并执行绘图操作，实现相关的渲染功能，在所有组件完成 update()之后，再进行组件 render()；
   e) 不同物体上的组件彼此调用：将被调用者挂载的物体作为调用者组件的初始化参数传输至调用者，由调用者通过物体获取被调用者示例进行调用；
5) 材质（metarial）：实现着色器控制器、着色器封装和纹理封装
   a) 着色器控制器：控制所有的着色器，并实现 uniform buffer 在各个着色器中的共享，使用手动生成的 metadata（./env/shader/controller 目录下）用于着色器控制器的初始化，着色器控制器的 metadata 中将存储
      i. 着色器 metadata 的路径；
      ii. uniform buffer 以及绑定点；

iii.uniform block 以及绑定点；

纹理控制器在运行时将被实例化为全局单例，控制所有着色器。

b) 着色器：封装 OpenGL 对着色器的各个操作，包括传送 uniform 数据、传送纹理数据、着色器源文件的编译和链接。使用手动生成的 metadata（./code/env/shader/**目录下）用于着色器的初始化，着色器控制器的 metadata 中将存储

i. 顶点着色器、几何着色器、片段着色器源文件路径；

ii. layout 数据；

iii.着色器中自定义的结构体；

iv. uniform 块的内容；

c) 纹理：封装 OpenGL 对纹理的各个操作，包括纹理的加载、纹理的配置、纹理的绑定。

6) 模型（model）：实现模型加载和模型渲染

a) 实现方式：使用开源库 Assimp 将模型加载为若干网络面片，保存位置、法向量、纹理坐标等内容用于模型渲染。

7) 渲染引擎封装文件（./code/src/public/engine 目录下）

a) 封装了鼠标输入（鼠标位置、鼠标相对与上一帧移动的距离、鼠标滚轮输入、鼠标左右按键输入）、渲染屏幕大小（长度、宽度、屏幕的实例）和渲染时间（当前时间、帧间隔）。

8) 场景生成代码（./code/src/public/scene 目录下）

a) 该代码将根据需要生成物体树（实际上生成了不同的根物体），构建出需要的场景，并实现场景之间的切换，本次大作业中的场景通过函数 makeWorkPieceScene()（line102 in scene.cpp）生成。

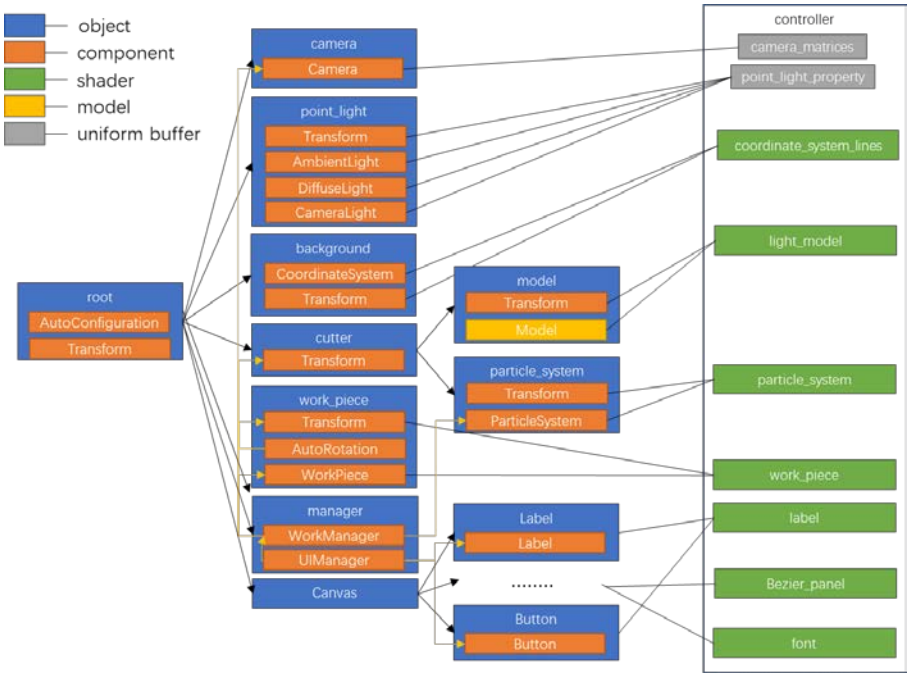
9) 自定义工具（./code/src/util 目录下）

a) 开源的 stb_image 图片加载工具；

b) 工件轮廓约束工具（包括 Bezier 曲线约束）；

3. 大作业程序实例结构

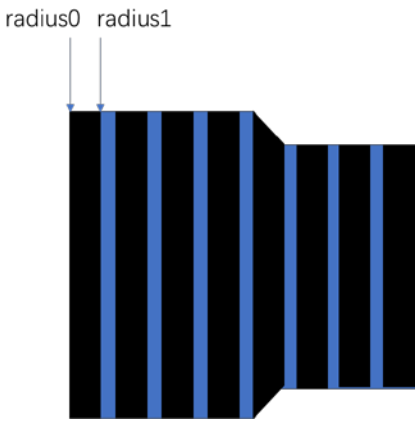
1) 结构图如图所示



2) 各个物体（object）实现的功能如下
   a) 根物体（root）：组织物体树、刷新颜色缓存
   b) 相机（camera）：实现相机功能
   c) 点光源（point_light）：为场景提供点光源
   d) 背景（background）：为场景提供坐标轴背景
   e) 刀具（cutter）：刀具模型位置控制
   f) 模型（model）：刀具模型的渲染
   g) 粒子系统（particle_system）：实现切削时产生的粒子效果
   h) 工件（work_piece）：圆柱体工件模型的建模及渲染
   i) 管理员（manager）：UI 界面控制和其他各个组件的控制
   j) UI 界面（canvas）：实现 UI 界面

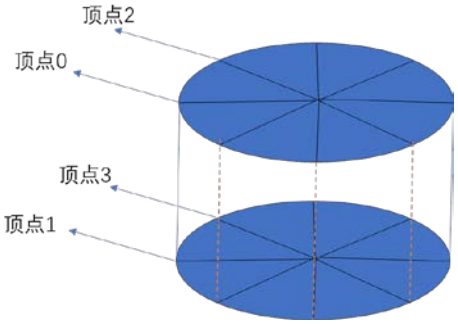
三、各个功能具体的实现方式

1. 圆柱状材料的几何建模及显示、刀具模型显示、背景显示、材料旋转动画
   1) 圆柱状材料几何建模（./code/src/component/work_piece 目录下）
      a) 使用微元法，将圆柱体切片，每一片都是一个小圆台，按照切片逐个记录各个切片点的半径即可得到每一个小圆台的上下半径，小圆台的高度是切片的宽度。
      b) 示意图



      c) 在实际实现过程中，蓝色部分的宽度为 0（即两个圆台之间彼此紧密贴合），黑色部分的宽度为切片单元的宽度。
      d) 当对工件进行切削时，直接修改切削位置记录的圆台半径信息即可。
      e) 允许用户通过 UI 界面调整材料的材质、半径和长度，详细信息位于 4.1.3。
      f) 备注：在后文中，圆台的上下表面指圆台的圆面，圆台的侧面指圆台的非圆面。
   2) 圆柱状材料的显示（顶点数据的构建位于./code/src/component/work_piece 和着色器代码位于./code/env/shader/work_piece）
      a) 将整个圆柱状的工件材料拆分成多个小圆台逐个渲染，所有小圆台公用同一个顶点数据（包括 VAO、VBO、EBO），在渲染过程中，组件通过 uniform 数据将半径数据和位置索引数据（该圆台位于圆柱体的哪一片）传输给着色器，由着色器将同一顶点数据渲染成不同大小不同位置的多个小圆台，在渲染过程中根据顶点的位置确定顶点的纹理坐标。
      b) 单个小圆台顶点数据的构建方法
         i. 顶点数据信息（VBO）：小圆台的顶点数据只需要记录侧面的顶点信息，

侧表面渲染和上下表面渲染共用同一套顶点数据，将小圆台的圆面的圆心为中心，将整个小圆台划分为 360 份，一共将产生 720 个顶点，其中每一个顶点包含 6 个浮点数据（3 个用于位置信息，3 个用于法向量），示意图如下



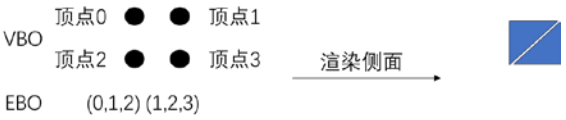
ii. 顶点索引信息(EBO)：为了将侧面和上下表面拆分成逐个三角形面片，需要对已有的顶点建立索引，按照“0,1,2”、“1,2,3”、“2,3,4”…“719,720,0”、“720,0,1”的逻辑方式建立共 720 个索引，占用 2160 个浮点数空间。

iii. 建立顶点数组对象（VAO）：将 VAO 与相应的 VBO 和 EBO 绑定即可。

c) 小圆台侧面渲染

i. 建立的顶点索引信息（EBO）包含的所有顶点构成的三角形面片的集合就是小圆台的侧面，在渲染时直接根据顶点数据和顶点缩影信息渲染即可。

ii. 渲染过程示意图



iii. 侧面顶点的纹理坐标

通过顶点的位置信息可以确定顶点的纹理坐标，将整个圆柱的侧面展开为一个长方形用于渲染纹理，例如对于长度为 3、半径为 0.5 的圆柱体，在切片位置为 1.5，所处角度为 90 的顶点，其纹理坐标为 texcoord =（1.5/3.0, 90/360）

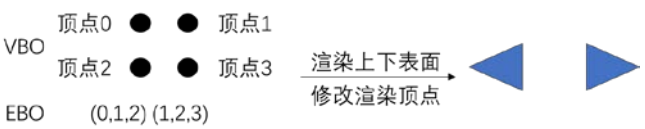
iv. 侧面顶点的法向量

侧面顶点的法向量仅与顶点所处的位置有关

d) 小圆台上下表面的渲染

i. 小圆台的上下表面渲染时将通过几何着色器对渲染顶点进行修改，将原本侧面的三角形面片的索引转化为上下表面的三角形面片的索引，例如原本的“0,1,2”索引，将被转化为“0,上表面中心,2”索引（由于索引奇偶性相同时对应点位于同一侧），原本的“1,2,3”索引，将被转化为“1,下表面中心,3”索引。

ii. 渲染过程示意图



iii. 上下表面顶点的纹理坐标

将上下表面的矩形纹理作为上下表面的外接矩形，根据顶点所处的角度确定其纹理坐标，例如处于 90 度的顶点纹理坐标为

texcoord =（cos0 * 0.5 + 0.5， sin90 * 0.5 + 0.5）

iv. 上下表面的法向量

上下表面的法向量默认为（1,0,0）或（-1,0,0）。

e) 圆柱状材料的纹理

i. 圆柱状材料的纹理有四种：木质切削前纹理，木质切削后纹理，铁质切削前纹理，铁质切削后纹理。通过加载不同的图片实现不同的纹理，使用 uniform 数据控制着色器选择不同的纹理。

f) 圆柱状材料的光照效果与材质

i. 使用点光源作为光源，使用 Blinn-Phong 模型模拟光照效果。

ii. 点光源的属性

- 位置（*position*）；
- 衰减属性（*constant,linear,quadratic*）；
- 环境光照（*ambient*）；
- 漫反射光照（*diffuse*）；
- 镜面光照（*specular*）；

iii. 物体材质属性

- 环境光照反射率（*ambient*）；
- 漫反射光照反射率（*diffuse*）；
- 镜面光照反射率（*specular*）；
- 材质光亮度（*shininess*）；

iv. 环境光照计算公式

$$\boldsymbol{ambient} = pointLight.ambient * material.ambient$$

v. 漫反射光照计算公式

$$normal = normalize(vertex.normal)$$
$$light\_dir = normalize(pointLight.position - vertex.position)$$
$$diff = \max(dot(normal, light\_dir), 0.0)$$
$$\boldsymbol{diffuse} = diff * pointLight.diffuse * material.diffuse$$

vi. 镜面光照计算公式

$$normal = normalize(vertex.normal)$$
$$view\_dir = normalize(camera\_pos - vertex.position)$$
$$halfway\_dir = normalize(light\_dir + view\_dir)$$
$$sp = pow(max(dot(norm, halfway\_dir), 0.0), material.shininess)$$
$$\boldsymbol{specular} = sp * pointLight.specular * material.specular$$

3) 刀具模型显示（./code/src/model 目录下）

a) 使用开源库 Assimp 加载刀具模型（./code/env/model/SVHBL163C.STL）。

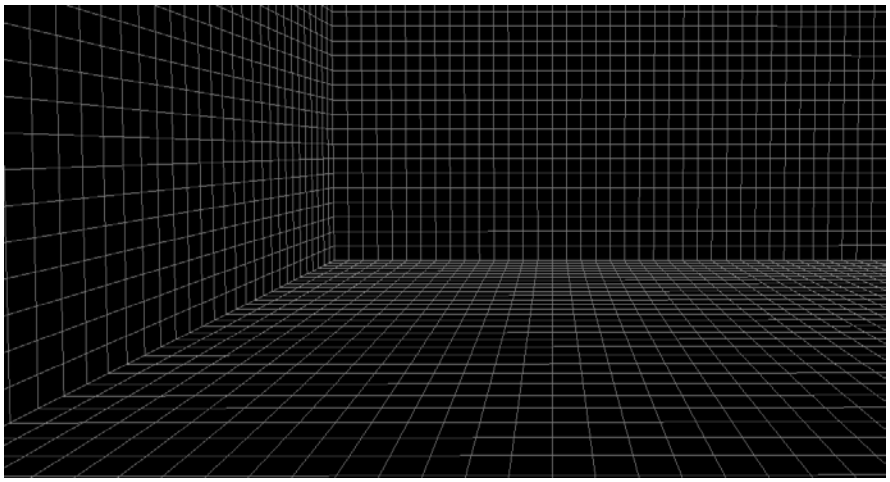
b) 在物体树上，通过 model 物体加载模型，基于 Transform 组件的继承关系，通过 cutter 物体控制 model 物体的位置和旋转角。

c) 加载的刀具模型会受到光照效果的影响，使用 Phong 模型模拟，原理与 Blinn-Phong 模型相似。

4) 背景显示（./code/src/component/CoordinateSystemLines 目录下）

a) 在 XoY 平面绘制若干条平行与 X 轴的线段和平行于 Y 轴的线段，线段之

间的间隔为 0.1，YoZ 平面、XoZ 平面同理，背景显示效果如下图所示



5) 圆柱材料旋转动画（./code/src/component/AutoRotation 目录下）
   a) 通过“自动旋转”组件控制圆柱材料的旋转，该组件会读取并修改圆柱材料的 Transform 信息。
   b) 得益于模块化设计，实现自动旋转的代码与第一次作业中使茶壶旋转的代码相同。

2. 鼠标控制刀具移动，原料模型产生变形，切削前后的光照效果变化
   1) 鼠标控制刀具移动（./code/src/component/WorkManager 目录下）
      a) 在鼠标左键按下后，通过刀具（cutter）的 Transform 属性可以获取刀具当前的位置，再通过相机的投影矩阵（projection）和视角矩阵（view）计算出此时刀具对应的标准化设备坐标，与光标的标准化设备坐标进行比较即可得到当前光标点击的位置是否为刀具所在的位置。
      b) 之后按住鼠标左键进行移动，刀具将按照一定的速度移动向光标。
      c) 为了更加真实地模拟切削，提高用户使用体验，实现了以下优化
         i. 在未切削物体时刀具的移动速度将比切削时更快；
         ii. 刀具一次能够切削的长度将收到限制（在演示版可执行文件中，为了方便演示，取消了该限制，在提交版和源代码中，保留了该限制）；
         iii. 刀具碰到 Bezier 曲线约束时，如果光标所在的方位将使得刀具穿过约束曲线，刀具不会立即停止移动，而是尝试轻微地修改移动的方向，实现更加顺滑的切削（否则需要严格地移动光标至 Bezier 曲线的切线方向，更难切削出需要的曲线，用户操作体验也很差）；
   2) 原料模型产生变形（./code/src/component/WorkManager 目录下）
      a) 在刀具移动的过程中，WorkManager 组件将读取刀具的位置，将刀具位置于预先记录的工件约束信息（详见 3.4.1）进行比较，将切削后的长度传递至工件，由工件修改自身记录的半径信息，实现变形、纹理修改和光照效果修改。
   3) 不同材质切削前后的光照效果不同（./code/src/component/WorkPiece 目录下）
      a) 在 3.1.2 中，已经详细说明了材质对光照效果的影响，此处不再赘述，材质的变化以小圆台为单位，每一个小圆台的材质都不同，小圆台的上下表面材质和纹理与切削后的侧面相同，切不随着切削过程变化。
      b) 切削前后木质或铁质的材质属性

i. 切削前木质材质

```
{
    "ambient": 0.5,
    "diffuse": 0.3,
    "specular": 1.0,
    "shininess": 2.0
}
```

ii. 切削后木质材质

```
{
    "ambient": 0.7,
    "diffuse": 0.5,
    "specular": 1.0,
    "shininess": 8.0
}
```

iii. 切削前铁质材质

```
{
    "ambient": 1.0,
    "diffuse": 1.0,
    "specular": 1.0,
    "shininess": 2.0
}
```

iv. 切削后铁质材质

```
{
    "ambient": 1.0,
    "diffuse": 1.0,
    "specular": 1.0,
    "shininess": 16.0
}
```

3. 刀具移动前端的粒子系统模拟碎屑飞溅

1) 粒子系统（./code/src/component/ParticleSystem 目录下）

a) 粒子系统开始/暂停控制

粒子系统对外暴露开始产生粒子和暂时产生粒子的接口，由 WorkManager 控制粒子系统是否产生粒子。

b) 粒子系统的位置控制

粒子系统组件所属的物体将是刀具物体的子物体，由此可以控制粒子产生的位置一定位于刀具前端。

c) 粒子系统的性能

由于粒子系统实现很简单，计算量不大，粒子系统无论开启和关闭，程序运行的帧数都是 60FPS，在 1000 粒子的情况下，CPU 占用小于 1%，在 100000 粒子的情况下，CPU 占用小于 5%。

d) 粒子系统的属性

i. 粒子的最大存在时间；

ii. 稳定时粒子的总数；

iii. 粒子发射的方向和角度偏差上限；

iv. 粒子的初速度和速度偏差上限（标量）；

v. 粒子的加速度；

vi. 粒子的颜色；

e) 单个粒子的属性

i. 粒子的位置；
ii. 粒子的速度；
iii. 粒子的加速度；
iv. 粒子的颜色；
v. 粒子已经存在的时间；

f) 粒子系统的生命周期
i. 在每次更新时，如果需要产生粒子（由 WorkManager），粒子系统将执行产生粒子的操作，无论是否需要产生粒子，粒子系统都会完成已有粒子的渲染和销毁。将同时帧内产生的粒子作为一个粒子集合，粒子集合是粒子系统产生粒子、更新粒子、渲染粒子和销毁粒子的基本单位。
ii. 粒子的产生
- 粒子产生数量的确定
  通过下面的公式可以计算出一段时间内被销毁的粒子数量，以此确定需要重新产生的粒子数量，以此保证稳定时粒子总数在预先确定的总数附近

  *粒子产生数量=帧间隔/粒子最大生存时间*稳定时粒子的总数*
- 粒子初始速度的确定
  生成 0 至 1 之间的随机数，使用下面公式计算出粒子的随机初速度

  粒子实际初速度=随机数*速度偏差上限+粒子的预设初速度
- 粒子初始角度的确定
  实际上通过预设方向和允许偏差的角度实际上将发射出的粒子限制在了一个锥形的范围内，需要随机产生一个在该范围内的一个方向

  随机法向量=cross(预设方向，任意随机方向)
  *粒子实际方向=预设方向以随机法向量为轴偏转随机数*角度偏差上限*

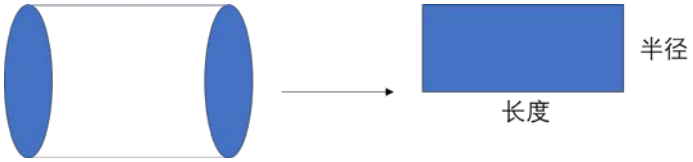
iii. 粒子的更新
- 使用 EulerExplicit 微元方法简单地计算粒子的在速度和加速度影响下的位置变化

  *下一帧位置 = 当前位置 + 当前速度 * 时间间隔*
  *下一帧速度 = 当前速度 + 当前加速度 * 时间间隔*

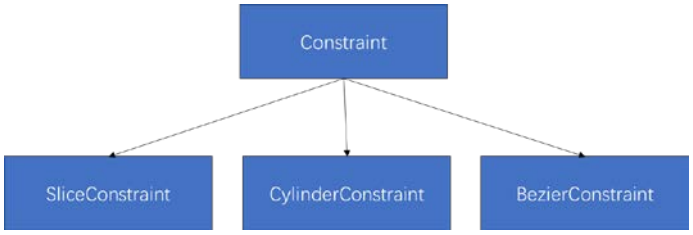
iv. 粒子的渲染
- 粒子系统将一个粒子集合中所有粒子的位置，颜色和速度传送至渲染管线中，在几何着色器中，粒子将由一个点按照粒子的速度方向扩展成一条很短的线段，实现更加真实的渲染效果。

v. 粒子的销毁
- 每次更新时，粒子系统都会检查粒子集合的存在时间，如果超出了存在时间将销毁整个集合的粒子。

4. 三次 Bezier 曲线交互创建及放置，作为切削交互的轮廓约束
   1) 切削约束实现（./code/src/util/Constraint）
      a) 由于所有的切削操作都被限制在中轴线上进行且上下对称，所有的切削约束都是针对工件的中轴线界面的一半进行设计，示意图如下



      b) 一个工件的约束实际上分为上界（即当前各个长度对应的半径）和下界（即添加的切削约束）两个部分，切削过程中如果刀具的预测位置大于上界，说明刀具没有进行切削，如果刀具的预测位置位于上界和下界之间说明刀具正在切削，需要实时修改上界，如果刀具的预测位置小于下届说明刀具即将超过下届约束，需要停止刀具移动。
      c) 根据需要，抽象出了三种约束，分别为切片约束（SliceConstraint），圆柱约束（CylinderConstraint）和 Bezier 约束（BezierConstraint），三种约束的关系如图



      i. 约束基类的属性
         - 采样点数组；
         - 长度区间；
         - 计算给定长度下半径的函数；

      ii. 切片约束
         - 作用：作为上界记录切削过程中各个片元的半径；
         - 初始化：通过给定的长度区间、半径、和采样间隔初始化，使用采样点数组存储长度区间内每一段的半径；
         - 计算给定长度下半径的函数：由于采样点遍历了整个长度区间，直接返回最接近给定长度的点记录的半径即可；
         - 切片约束允许创建后修改，便于实时修改上界；

      iii. 圆柱约束
         - 作用：该约束实际并未使用；
         - 初始化：通过给定的长度区间、半径、初始化，使用采样点数组存储长度区间内每一段的半径；
         - 计算给定长度下半径的函数：直接返回记录的半径值；

      iv. Bezier 约束
         - 作用：作为下界辅助切削成需要的曲线；
         - 初始化：通过给定四个 Bezier 控制顶点初始化（此处没有检查控制顶点的有效性，控制顶点的有效性检查在 3.4.2 中进行），使用参数函数表示 Bezier 曲线，将 Bezier 曲线参数由 0 到 1 以 0.001 为间隔变化，根据给定的控制点采样得到 1000 个采样点；

- 计算给定长度下半径的函数：在 1000 个采样点中，通过二分查找法寻找到最接近的两个采样点，进行线性差值，得到给定长度在 Bezier 曲线约束下的半径；

2) 三次 Bezier 曲线交互创建及放置（UI 界面控制组件位于./code/src/component/Manager，UI 界面显示组件位于./code/src/component/UI）

a) 使用 UI 界面实现 Bezier 曲线的创建和放置，UI 界面示意图如下

b) 控制点聚焦

通过点击左侧的控制点下方的四个数字按钮可以选择当前聚焦的控制点，默认聚焦控制点 0，被选择的控制点按钮将更暗，每次选择控制点都会在日志窗口显示相应的信息。

c) 约束显示窗口

i. 在 Bezier 约束窗口中，白色的子窗口为约束显示窗口。

ii. 约束显示

- 使用黑色线表示原圆柱工件的形状；
- 使用红色线表示约束上界（即实时切削后的形状）；
- 使用绿色线表示已经添加的约束下界；

iii. 控制点放置和修改

- 在约束窗口中点击，将使用蓝色的点显示当前聚焦的控制点的位置，并在日志窗口显示更加详细位置信息；
- 当确定四个控制点的位置后，将检查控制点的有效性（要求所有控制顶点按照索引顺序从左至右依次排列），如果控制点有效，将使用蓝色线表示临时的 Bezier 曲线，此时约束不会生效；
- 当生成临时的 Bezier 曲线后，可以点击应用按钮，将 Bezier 曲线添加至下界约束，此时约束已经生效；

四、其他程序相关内容

1. 程序使用说明

1) 运行可执行程序

在./code/env 目录下存放了运行所需的 dll 文件，着色器文件，材质文件，模型文件，运行提交版.exe 或演示版.exe 即可。

2) 编译源代码

使用 Visual Studio 2019 打开./code 目录下的 VS2019_demo.sln 文件，使用 X64-Release 模式编译即可，项目中已经配置了编译生成的 exe 文件的运行目录为./code/env，按下 F5 即可在自动生成运行。

3) 操作说明

a) 模式切换：通过 E 键可以切换移动模式与切削模式，在移动模式下，光标会被隐藏，在切削模式下，可以移动刀具并操控 UI 界面。

b) 退出按键：按下 Esc 键即可退出。

c) 相机移动：在移动模式下，通过 WASD 键可以控制相机的前后左右移动，通过左 Shift（下降）和空格（上升）可以控制相机的上下移动。

d) 刀具移动：在切削模式下，可以通过按住鼠标左键鼠标控制刀具对工件进

行切削，在切削过程中，刀具会按照一定的速度移动向光标，但是如果出现了不合理的切削，刀具会停止移动。

e) UI 界面：控制工件类型、长度、半径，控制 Bezier 曲线约束

i. UI 界面图如图所示

ii. 在工件配置窗口，可以配置工件的材质（木质或铁质）、工件的长度（最小为 3，最大为 6）、工件的半径（最小为 0，最大为 0.5），每次修改配置都会在最右侧的日志窗口显示当前操作的详细信息。

iii. 在 Bezier 约束窗口，具体的操作方式在 3.4.2 中已经详细说明

2. 程序编程环境

1) 电脑硬件

a) CPU：Ryzen R5 3600；

b) 内存：十栓 16GB * 2；

c) 显卡：耕升 RTX 2060；

d) 硬盘：西部数据 SN550；

2) 操作系统：Windows 10 专业版

3) IDE 及编译器：Visual Studio 2019（自带 MSVC 编译器）

4) 默认编译配置

a) 所有库文件及源文件均使用 X64-Release 方式编译

5) OpenGL 相关库文件：

a) glfw（3.3.2），编译成静态链接库，运行时会调用动态链接库；

b) glad（3.3 core），将头文件及源文件直接添加至项目中；

6) 其他使用的库

a) glm（0.9.9），数学计算库，使用时将头文件直接添加至项目中；

b) assimp（5.0.1），模型加载库，使用时编译成静态链接库，运行时会调用动态链接库；

c) JsonCpp（1.y.z），Json 文件解析库，使用时编译成静态链接库；

d) freetype（2.10.4），文字加载库，使用时编译成静态链接库，运行时会调用动态链接库；

e) stb_image（2.26），图片加载库，使用时将头文件直接添加至项目中；

7) 部分参考代码

a) 模型加载

b) 字体显示