浙江大学

计 算 机 图 形 学
Computer Graphics

课 程 大 项 目 报 告
Project Report

1. 姓名：王方懿康　学号：3190101086　电话：13357173236

2. 姓名：章沈柯　学号：3190101091　电话：18767196696

3. 姓名：钟沛沛　学号：3190102111　电话：17799857686

4. 姓名：徐梓凯　学号：3190103058　电话：19883500086

5. 姓名：张立昱　学号：3190105131　电话：15830420312

6. 姓名：包德政　学号：3190105240　电话：19817866048

https://github.com/snoopy10086/CGProject

指导老师：张宏鑫教授

2021 - 2022秋冬学期　2022 年 1 月 29 日

目 录

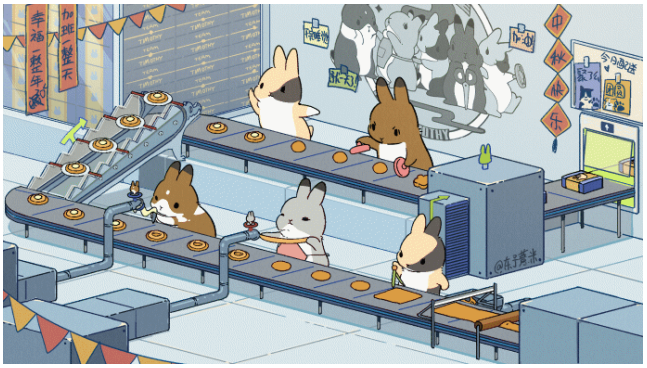
1 程序概述

1.1 选题背景

1.1.1 选题概述

根据课程项目要求，本项目应以工业智能为主题，采用 C++编程语言，并基于 OpenGL API 构建一个较为完整的三维 CG 世界。为同时保证课程项目选题的技术性、创新性与可玩性，开发小组结合网络热点：兔子的月饼工厂（作者新浪微博：@东予薏米），确定了选题内容。

结合背景创意，我们的项目讲述了这样的一个故事：时值金秋，中秋佳节即将来到，可爱的小兔子们正在努力为按时交付月饼礼物而辛勤加班。月饼制造的过程可以分为馅料与饼皮结合、成型月饼包装两个部分，请建造一个智能工厂，帮助小兔子们吧！“加班一整天，幸福一整年”！



1.1.2 选题内容

基于 OpenGL API， 使用 C++语言构建一个月饼制造的流水线工厂。

在基本布局方面，实现基本的工厂设施、基本体素、环境布局与可控光照等效果；在视觉效果方面，实现漫游移动、实时渲染等效果；在生产效果方面，在基本的生产水平基础上，添加智能化设备，实现局部的自动化运行等效果；在运行效果方面，实现运行流畅、可控截图、动画效果等功能；在用户体验方面，实现用户对场景进行局部 DIY 等效果。

1.1.3 技术性

作为一个较为完备的流水线生产系统，完全覆盖了《CG 课程大作业 2021》的要求，并基本涉及了《计算机图形学》课程所讲授的理论内容与实验内容，包括：基本编程管线、动画效果、光照、纹理、鼠标键盘响应以及漫游效果等。同时在实现选题内容的过程中，也涉及了多文件编程、基于 github 的多开发者协作、利用开放文档交流合作等技术内容。

1.1.4 创新性

相较传统的智能工厂，本项目选题更加贴近当下网络热点，更加具有趣味性与中国文化韵味。

程序中多处进行了面向使用者考虑的趣味性设计，比如多种口味的月饼、多种款式的礼物盒以及可自由开合的门窗等。

而基于月饼制造的智能工厂的选题，也考虑到了中国文化元素在程序中的体现。

我们的程序是基于真实世界模拟的，不存在 Minecraft 游戏那样的像素思想，也就是说我们的全部游戏世界是连续的而非离散的，代码实现难度更大，效果更贴近现实。

总之，无论是开发者还是使用者，均能在本项目中体验到技术开发带来的趣味，品味元宇宙中蕴藏的中国风。

1.1.5 可玩性

选题规划除提供一完整的流水线 demo 外，还支持用户进行多种拓展，包括环境光源与环境设施的变动，基本材料的添加、修改与移动，在流水线外自行组合碰撞合成物质，以及基于文件的数据导入导出功能等。整体上满足了用户基本的 DIY 需求，增强了可玩性。

1.2 目标要求

基于 OpenGL API，采用 C/C++，构建一个完整的 CG 世界。

项目基本目标为：

A. 以工业智能为主题；

B. 实现一个三维场景建模与真实感绘制的小系统；

C. 具有场景编辑与执行两种模式；

D. 完成全部的基本需求。

项目附加目标为：

A. 基于 Github 进行开发，使开发流程清晰可见，减轻不必要的麻烦；

B. 代码编写应遵守编程规范，要求有适当注释，具有良好的可读性与可拓展性；

C. 合理规划时间，统筹小组合作，适量完成 Bonus 需求。

1.3 术语说明

术语	说明
机器人/机械臂	指具备抓取、移动的自动化、智能化机械臂装置。
传送带	指具备单向传送物体功能的带状机械装置。
告示板	指显示选题内容、可变换内容的木质结构。
月饼馅料	指可供合成月饼的基本球体材料，以颜色区分馅料。

礼物盒皮	指可供合成月饼盒的、未折叠的纸板结构，以颜色区分类型。
光源色温	指根据色温-RGB 对应表得到的不同色温对应的颜色光源显示效果。

2 效果展示

2.1 环境介绍

工厂所处环境是一长方体构成的车间。

车间上方共设置 16 （4*4）盏照明灯，其中处于对角线上的 8 盏可被点亮，车间上方中央设置有一自动旋转的风扇。

车间四壁共设置有三扇窗与两扇门，均可自由开闭。

车间主体空间由一玻璃板划分为两个大小不等的空间，较小的空间设有若干书柜与一告示板，较大的空间设有流水线 demo。流水线 demo 主要由传送带、机械臂及物体放置台组成。

2.2 快捷键说明

视角变换

- `w` `s`：视角方向前后移动
- `a` `d`：视角方向在水平平面的投影的法线方向移动，即在同一个水平面里动
- `z` `c`：上下动

碰撞检测

- `x`：视角移动时会/不会检测碰撞；

机械臂操作

- `i`：控制机械臂 1 传送；
- `o`：控制机械臂 2 传送；
- `k`：控制机械臂 3 传送；

- `l`：控制机械臂 4 传送；

机械爪操作
- `e`：控制三个机械臂的机械爪抓取；

传送带操作
- `r`：控制传送带 1&2 号转动（按一下动一下）；
- `t`：控制传送带 3 号转动（按一下动一下）；
- `y`：控制传送带 4 号转动（按一下动一下）；
- `u`：控制传送带 5 号转动（按一下动一下）；

门窗控制
- `f` `g` ：控制两扇门的开与关; 其中`f`为传送带方向的门的开关，在开门时，如果传送带 5 末端有**礼物盒**，则礼物盒消失。
- `v` `b` `n` ：控制三扇窗开与关；

光源控制
- `1` `2` `3` `4` `5` `6` `7``8`：分别控制 0-7 号光源的开与关（视觉效果为灯的开关）；
- `9`：控制光的色温循环变化；

公告板纹理变换
- `0`：改动公告板;

截图操作
- `p`：按下截图（当前画面）

退出程序
- `q` `ESC`：退出程序

导入导出
- `K` ：存储数据
- `L` ：导出数据

选中物体操作
- `+` ：放大物体
- `-` ：缩小物体
- `A` ：右移物体
- `D` ：左移物体
- `Z` ：上移物体
- `C` ：下移物体
- `S` ：前移物体
- `W` ：后移物体
- `Backspace` ：删除物体
- `h` ：切换创造月饼合成类 shape 创建还是其他 shape 的创建
- `j` ：在同一 shape 创建模式下切换不同的 shape，如在月饼、月饼皮、馅料间切换

2.3 Demo 说明

1、相机漫游：

智能工厂可以通过移动鼠标移动调整视角的欧拉角，并且利用键盘按键 awsdzc 进行相机移动。相机漫游有两个模式，一类似于创造模式，不受物块阻挡；二是观看者模式，相机被限制在非生产区域，同时会进行实时碰撞检测。使用 x 可以切换两种模式。

2、工厂环境：

工厂被划分成两个部分，分别是生产区域和非生产区域。放置传送带、机械臂的生产区域中，可以将原材料合成礼物盒并且输送到门外；非生产区域则有柜子、木桌和公告板，其中公告板的样式可以通过 0 来更换。

3、门窗表达：

工厂有两扇门，分别位于生产区域和非生产区域，利用 f、g 控制开闭。墙面上的三扇窗则使用 v、b、n 进行控制。

除此之外，工厂中有利用 Nurbs 曲面绘制的电风扇，在生产过程中自动旋转，下方有被风吹动的条带。

4、光源开闭：

天花板上有 4x4 个灯泡，考虑到光源数量限制，在对角线上设置为聚光灯，也有环境光，可以通过 1-8 单独进行光源开闭，并且利用 9 修改色温。

5、物体控制：

单击非传送带区域时，有两种可能。一是点击位置与其他物体重合，则会选中这一物体，体现于物体变为框架结构，并且上方出现箭头指明。这些物体可以利用-、+修改大小和利用 AWSDZC 修改坐标，并且使用 delete 来删除物体。如果点击位置没有其他物体，则进行物体放置。放置物体有两种模式，分别是月饼相关原料和物块，利用 h 来进行模式切换，同一模式的不同物体则使用 j 进行切换。

6、传送合成：

本工厂的目的是利用月饼原料，即馅料、月饼皮和包装盒合成出一个包装好的月饼，并将其传送到屋外出口。demo 中各传送带上已经放好了原料，使用 r、t、y、u 进行传送。当一个原料抵达传送带重点时，可以使用机械臂跨传送带运输，机械臂使用 i、o、k、l 操控。不同口味的馅料可以生成不同口味的月饼，一共有抹茶、红豆和香芋三种口味；不同颜色的礼物盒纸板也可以生成不同的礼物盒，有两种颜色。

7、截屏保存：

快捷键 p 可以进行截屏保存，将当前的页面保存为 bmp 文件，以 CGProject_当前时间.bmp 命名。

8、物体保存与读取：

快捷键 K 可以进行场景内物体保存，将当前的场景内物体保存为文本文件，命名为 Shape.ZJUCG。

快捷键 L 可以进行文件物体读取，将 Shape.ZJUCG 文件内存储的信息还原到场景内。

2.4 DIY 介绍

wasd 控制相机移动, zc 控制相机升降, 鼠标控制视线方向, q 键退出

1. 相机漫游: 相机漫游采用类 FPS 自由视角, 鼠标移动会改变视角的欧拉角, 方向键前后会跟着视线方向移动, 左右会水平移动。

2. 鼠标选择拾取: 鼠标对准物块, 即可选中物块, 物块上方出现选中的箭头指示, 鼠标选中之后使用键盘操作可以修改对应的物块对象, 例如修改该物块对象的空间位置或者大小。
3. 鼠标点击放置: 鼠标对准空地时, 会在空地的位置放置一个物块对象
4. 灯光可定制化, 数字键更改光源选项
5. 可以通过键盘控制机械爪的抓取, 机械爪会检测身前是否存在物块进行抓取操作或者放置操作
6. 可以通过键盘控制传送带的运转, 传送带会带动在传送带上的物块的移动

2.5 特性介绍

1. 巧妙地将利用本学期所学习的基本图形学知识，发挥创造力与主观能动性完成了一个比较完整的工厂场景。如使用 nurbs 曲线构建的风扇，使用更换纹理图案的方式来表现传送带运动这一循环运动方式。
2. 交互良好，比如选中一个物体时，该物体会变成线框模式，且上方会有白色棱形进行指示。
3. 我们创造性地使用“碰撞检测”的方法来进行月饼生产，即检测到馅料和月饼皮在同一位置时，会在原位置变成月饼，表现月饼的生产；月饼碰撞礼物盒皮同理。用靠近传送带的门打开时，传送带末端的月饼礼盒消失表现礼盒被运出工厂。

2.6 要求对照

2.6.1 项目基本要求

序号	要求	是否实现	具体实现描述
1	建模：基于 OpenGL/WebGL，具有基本体素（立方体、球、圆柱、圆锥、多面棱柱、多面棱台）的建模表达能力	是	实现了基本类 Shape，在此之上继承实现了立方体类 Cube，球体类 Sphere，圆柱类 Cylinder，圆锥类 Cone，六棱柱类 Prism，六棱台类 Trustum，圆台类 ConeCylinder。实现了所有要求体素的建模。
2	存储：具有三维网格模型导入导出功能（建议 OBJ 或 DAE 格式）	是	通过键盘 K 来存储，键盘 L 来读取。存储对象信息包括月饼相关元素的位置、大小、朝向。文件夹下的 Shape.ZJUCG 文件（文本文件）是存储于硬盘的介质。
3	编辑：具有材质、纹理的显示和编辑能力	是	不同物体具有不同材质，可以显示不同的纹理；同一物体，例如公告板上的纹理样式可以通过键盘按键进行切换编辑。
4	变换：具有几何变换功能（旋	是	机械臂实现了较为复杂的旋转效

	转、平移、缩放等）		果（4 DoF）放置的物块也可以进行几何变换。
5	光照：光照明模型要求，并实现光源编辑（如调整光源的位置，光强等参数）	是	实现了光源位置、光源开关与光源颜色的编辑。
6	漫游：能对建模后场景进行漫游如 Zoom In/Out，Pan, Orbit, Zoom To Fit 等观 察功能	是	采用类 FPS 模式进行视角变换，可以利用鼠标改变视角的欧拉角，并且利用键盘进行相机漫游。
7	记录：能够提供动画播放功能（多帧数据连续绘制），能够提供屏幕截取/保存功能	是	可将屏幕截取并保存为 bmp 格式图片；

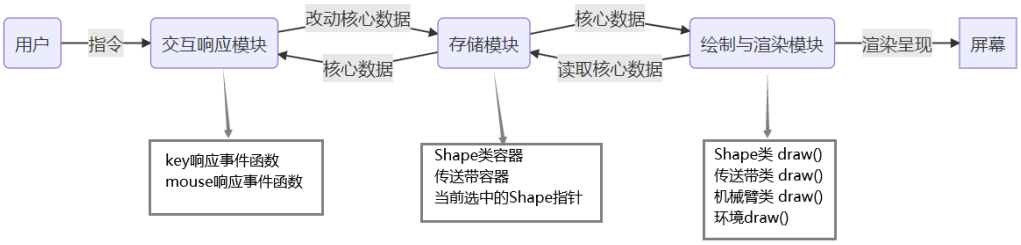
2.6.2 项目高级要求

序号	要求	是否实现	具体实现描述
1	具有 NURBS 曲面建模能力	是	电风扇采用 NURBS 曲面建模。
2	漫游时可实时碰撞检测	是	相机漫游时会进行碰撞检测，不会超出墙面。使用快捷键切换到观者模式后，会实时检测坐标，避免碰撞和穿模柜子、桌子等物体。
3	光照明模型细化，可任选实现实时阴影、Caustic、位移纹理、全局光照明（光 子跟踪）、辐射度、AO 叠加等	否	————
4	采用 HTML5/IOS/Android 移动平台实现	否	————
5	构建了基于此引擎的完整三维游戏，具有可玩性	是	构建由 5 个传送带和 4 个机械臂组成的默认流水线，玩家可通过键盘按键操纵对应的传送带和机械臂，将月饼皮和馅料合成 3 种不同月饼，将月饼和礼盒包装皮合成礼盒。
6	与虚拟现实 / 增强现实应用结合	否	————
7	具有一定的对象表达能力，能够表达门、窗、墙等	是	工程中实现了可开关的门、窗等对象，以及有自动旋转的电风扇和可进行流畅抓取物体的机械臂等物体。
8	复杂材质效果	是	工程中支持多重纹理映射、进行纹理编辑等功能。

3 程序设计

3.1 整体结构

整体软件架构采用交互-数据-显示模型，交互响应模块处理用户的操作后对存储模块中的核心数据进行修改，而每一帧内，绘制模块只会根据存储模块的核心数据信息来绘制场景。这样做的好处在于实现了交互与显示的解耦合，各模块的功能分工清晰，实现简单。



3.2 类与变量

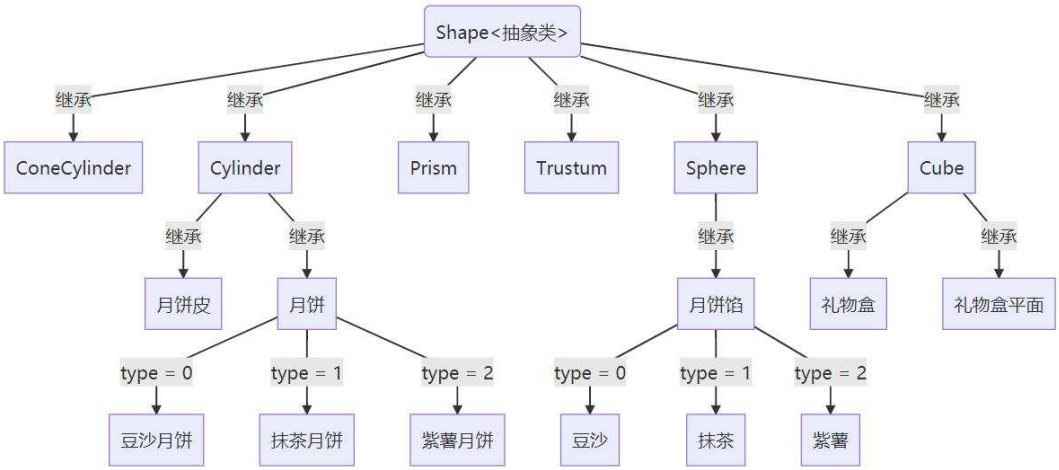
3.2.1 环境

无类设计。

3.2.2 光照

无类设计。

3.2.3 物体



类关系示意图如上，实现在 Shape.h 和 Shape.cpp 中。

类名	Shape
类的功能	基本类。定义了一个物体的位置坐标、旋转角度、缩放比例、纹理、类型，返回物体坐标信息、类型信息，对物体进行平移、旋转、缩放。

类变量	float globalX=0, globalY=0, globalZ=0; // 世界坐标系的坐标，并规定 **globalX、globalY、globalZ 是每个 Shape 底面中心坐标，便于后续操作，**（可以在每个 Shape 的 Draw 函数中通过 glTranslatef 等函数实现）。
	float rotateX=0, rotateY=0, rotateZ=0; // 相对于 x,y,z 轴旋转角度
	float scaleX=1, scaleY = 1, scaleZ = 1; //x,y,x 方向的缩放比例
	int Texture=0; //纹理
	int Texture2=0;
	int Type = -1; //标识这个 shape 的类型

类函数	double getGlobalX(); double getGlobalY(); double getGlobalZ();
	void transfer(double dx, double dy, double dz); // 平移
	void rotate(double dx, double dy, double dz); // 旋转
	void scaling(float scaleX, float scaleY, float scaleZ);
	virtual void Draw(); // 供外部调用的绘制函数
	int RetType();

类名	ConeCylinder
类的功能	圆台类，在 Shape 类上继承。实现任意大小圆台的绘制。
类变量	（新增的类变量，不包括父类） 无

类函数	（新增的类函数，不包括父类）
	public:
	virtual void Draw();
	ConeCylinder(float globalX, float globalY, float globalZ);
	private:
	void Texture_ConeCylinder(int i);

	void Texture_Circle(int i, float r);

类名	Cylinder
类的功能	圆柱类，在 Shape 类上继承。实现任意大小圆柱的绘制。
类变量	（新增的类变量，不包括父类） 无

类函数	（新增的类函数，不包括父类）
	public:
	virtual void Draw();
	Cylinder(float globalX, float globalY, float globalZ);
	protected:
	void Texture_Circle(int i, float r);
	void Texture_CylinderCircle(int i,int j);
	void Texture_Cylinder(int i);

类名	YueBingPi
类的功能	月饼皮类，在 Cylinder 类上继承。实现月饼皮的绘制。
类变量	（新增的类变量，不包括父类） 无
类函数	（新增的类函数，不包括父类） public: YueBingPi(float globalX, float globalY, float globalZ);

类名	YueBing
类的功能	月饼类，在 Cylinder 类上继承。实现 3 种不同月饼的绘制。
类变量	（新增的类变量，不包括父类） 无
类函数	（新增的类函数，不包括父类） public: YueBing(float globalX, float globalY, float globalZ); YueBing(float globalX, float globalY, float globalZ, int Type); protected: void SetType(int t);

类名	Sphere
类的功能	球体类，在 Shape 类上继承。实现 3 种不同月饼馅料的绘制。
类变量	（新增的类变量，不包括父类） 无

类函数	（新增的类函数，不包括父类）
	public:
	virtual void Draw();
	Sphere(float globalX, float globalY, float globalZ);
	Sphere(float globalX, float globalY, float globalZ, int Type);
	protected:
	void Texture_Sphere(int i);

类名	Cone
类的功能	圆锥类，在 Shape 类上继承。实现任意大小圆锥的绘制。
类变量	（新增的类变量，不包括父类） 无

类函数	（新增的类函数，不包括父类）
	public:
	virtual void Draw();
	Cone(float globalX, float globalY, float globalZ);
	private:
	void Texture_Cone(int i);

类名	Cube
类的功能	立方体类，在 Shape 类上继承。实现任意大小立方体的绘制。
类变量	（新增的类变量，不包括父类） 无

类函数	（新增的类函数，不包括父类）
	public:
	virtual void Draw();
	Cone(float globalX, float globalY, float globalZ);
	private:
	void Texture_Cone(int i);

类名	LiWuHePingMian
类的功能	礼物盒平面图类，在 Cube 类上继承。实现 2 种礼物盒平面图的绘制。
类变量	（新增的类变量，不包括父类） 无
类函数	（新增的类函数，不包括父类） public: virtual void Draw(); LiWuHePingMian(float globalX, float globalY, float globalZ); LiWuHePingMian(float globalX, float globalY, float globalZ, int Type);

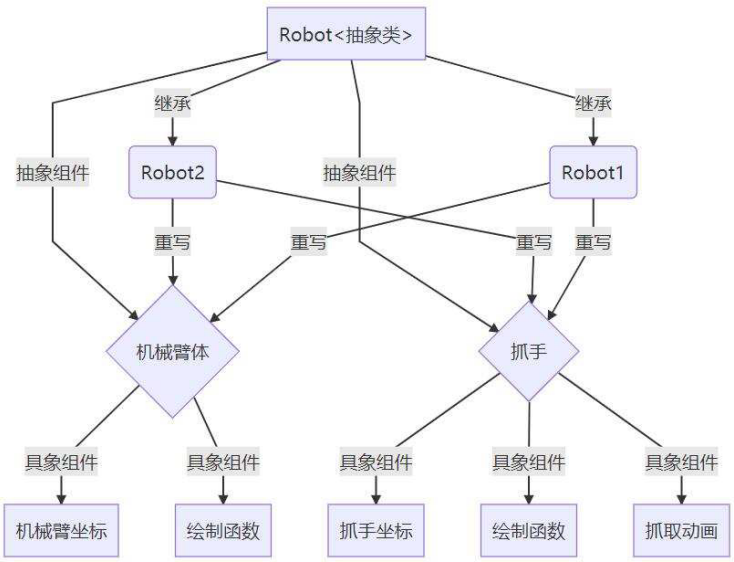
类名	LiWuHe
类的功能	礼物盒类，在 Cube 类上继承。实现 2 种礼物盒的绘制。
类变量	（新增的类变量，不包括父类） 无

类函数	（新增的类函数，不包括父类）
	public:
	virtual void Draw();
	LiWuHe(float globalX, float globalY, float globalZ);
	LiWuHe(float globalX, float globalY, float globalZ, int Type);
	private:
	void texture_LiWuHe(int i);

类名	Prism
类的功能	六棱柱类，在 Shape 类上继承。实现任意大小六棱柱体的绘制。
类变量	（新增的类变量，不包括父类） 无
类函数	（新增的类函数，不包括父类） public: virtual void Draw(); Prism(float globalX, float globalY, float globalZ);

类名	Trustum
类的功能	六棱台类，在 Shape 类上继承。实现任意大小六棱台的绘制。
类变量	（新增的类变量，不包括父类） 无
类函数	（新增的类函数，不包括父类） public: virtual void Draw(); Trustum(float globalX, float globalY, float globalZ);

3.2.4 机械臂



类关系示意图如上，实现在 Robot.h 和 Robot.cpp 中。

类名	Robot
类的功能	机械臂基类。定义机械臂在世界坐标系中的坐标，机械臂是否绑定物体以及、机械臂绑定的物体
类变量	/* the position of the robot */ float GripperX; float GripperY;

	float GripperZ; /* the bound shape */ Shape* TheShape;

类函数	public:
	virtual void Draw();
	virtual void update();
	protected:
	void Draw_Robot();
	void Draw_Claw(float angle);

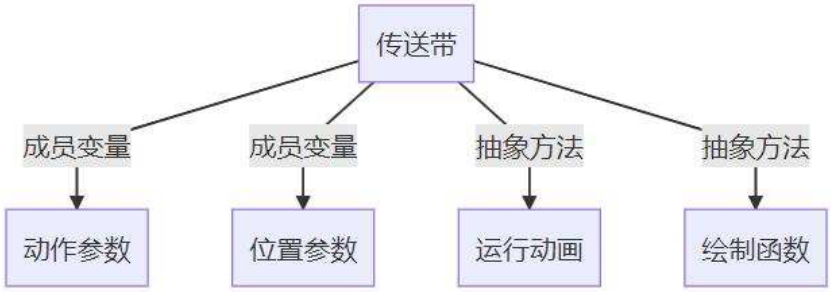
类名	Robot_2
类的功能	机械臂子类，继承 Robot 类。用于绘制除机械爪之外的其他部分，包括设置机械臂四个自由度旋转角度、机械臂整体大小、判断是否绑定、完成机械爪抓取动画等

类变量	/* four DoFs of the robot */
	GLfloat rotate1 = 90;
	GLfloat rotate2 = 0;
	GLfloat rotate3 = 45;
	GLfloat rotate4 = 90;
	/* whether the robot binds to a shape */
	bool IsBind = false;
	/* whether to check if bind */
	bool Check = false;
	/* the time flag of capture process animation */
	int timeflag = -1;
	float PositionX = 4;
	float PositionY = 0.1;
	float PositionZ = 4;
	/* overall size */
	float ZoomIndex = 1.0;
	Robot_2(float PositionX, float PositionY, float PositionZ);

类函数	protected:
	/* the methods used to draw the robot */
	void DrawJoint();
	void DrawRod(GLdouble baseR, GLdouble topR, GLdouble h);
	/* find the position of bound shape */
	void SetBindPosition();
	public:
	void Draw();
	void update();
	/* the grab animation */
	void HandleRotate();

	void setPositionX(float position);
	void setPositionY(float position);
	void setPositionZ(float position);
	void not_is_bind();

3.2.5 传送带



类关系示意图如上，实现在 conveyor.h 和 conve.cpp 中。

类名	conveyor
类的功能	传送带类。实现传送带的绘制，实现传送带运动效果，判断某个物体是否在传送带上。

类变量	float PositionX=0; // 世界坐标系的坐标
	float PositionY=0;
	float PositionZ=0;
	//并规定 Position**X**、Position**Y**、Position**Z 是每个传送带底面中心坐标，便于后续和物体的交互。**
	float MotionX = 0;//传送带运动参数
	float MotionZ = 0;
	int count = 0;//表示传送带运动时纹理切换

类函数	public:
	conveyor(float PositionX, float PositionY, float PositionZ, float MotionX, float MotionY);
	void rotate(double dx, double dy, double dz); // 旋转
	protected:
	void Texture_Circle(int i);
	void Texture_Cylinder(int i);
	void DrawConveyor2();//传送带具体绘制函数
	public:
	bool ifOntheConveyor(Shape* shape); //to judge if the shape is on this conveyor
	void AddMotion(Shape* shape); //使传送带上的物体运动
	virtual void draw();

3.2.6 其他

3.2.6.1 Shape 保存与读取

按键	大写 K
按键功能	保存当前场景中的月饼、月饼皮、礼物盒、馅、礼物盒平面的数据到 Shape.ZJUCG 文件下。数据包括位置、朝向、大小和品种。

```
case 'K':
  {
      ofstream outfile("Shape.ZJUCG", ios::trunc);
      vector<Shape*>::iterator Siter;
      Cylinder* s1 = new Cylinder(3, 0, 4);
      for (Siter = Shapes.begin(); Siter != Shapes.end(); Siter++) {
        if ((*Siter)->Stype == 1) {
          //outfile << " YueBing " << endl;
          outfile << "YueBing " <<
            (*Siter)->globalX << " " << (*Siter)->globalY << " " << (*Siter)->globalZ
            << " " << (*Siter)->rotateX << " " << (*Siter)->rotateY << " " << (*Siter)->rotateZ
            << " " << (*Siter)->scaleX << " " << (*Siter)->scaleY << " " << (*Siter)->scaleZ
            << " " << (*Siter)->Type << endl;
        }
        else if((*Siter)->Stype == 2){
          outfile << "YueBingPi " <<
            (*Siter)->globalX << " " << (*Siter)->globalY << " " << (*Siter)->globalZ
            << " " << (*Siter)->rotateX << " " << (*Siter)->rotateY << " " << (*Siter)->rotateZ
            << " " << (*Siter)->scaleX << " " << (*Siter)->scaleY << " " << (*Siter)->scaleZ
            << endl;
        }
        else if ((*Siter)->Stype == 3) {
          outfile << "Sphere " <<
            (*Siter)->globalX << " " << (*Siter)->globalY << " " << (*Siter)->globalZ
            << " " << (*Siter)->rotateX << " " << (*Siter)->rotateY << " " << (*Siter)->rotateZ
            << " " << (*Siter)->scaleX << " " << (*Siter)->scaleY << " " << (*Siter)->scaleZ
            << " " << (*Siter)->Type << endl;
        }
        else if ((*Siter)->Stype == 4) {
          outfile << "LiWuHePingMian " <<
            (*Siter)->globalX << " " << (*Siter)->globalY << " " << (*Siter)->globalZ
            << " " << (*Siter)->rotateX << " " << (*Siter)->rotateY << " " << (*Siter)->rotateZ
            << " " << (*Siter)->scaleX << " " << (*Siter)->scaleY << " " << (*Siter)->scaleZ
            << " " << (*Siter)->Type << endl;
        }
        else if ((*Siter)->Stype == 5) {
          outfile << "LiWuHe " <<
```

```
38.                 (*Siter)->globalX << " " << (*Siter)->globalY << " " << (*Siter)->globalZ
39.                 << " " << (*Siter)->rotateX << " " << (*Siter)->rotateY << " " << (*Siter)->rotateZ
40.                 << " " << (*Siter)->scaleX << " " << (*Siter)->scaleY << " " << (*Siter)->scaleZ
41.                 << " " << (*Siter)->Type << endl;
42.         }
43.         else {
44.             //outfile << " shapes " << endl;
45.         }
46.
47.     }
48.     outfile << "#";
49.     outfile.close();
50.     break;
51.   }
```

按键	大写 L
按键功能	读取当前目录下的 Shape.ZJUCG 文件，并根据文件中的记录来还原场景中的月饼、月饼皮、礼物盒、馅、礼物盒平面。

```
1.  case 'L':
2.    {
3.      //erase all the shapes
4.      Shapes.clear();
5.      //read in data stored
6.      ifstream myfile("Shape.ZJUCG");
7.      string str;
8.      float Gx, Gy, Gz;
9.      float Rx, Ry, Rz;
10.     float Sx, Sy, Sz;
11.     int type;
12.     myfile >> str;
13.     while (str.compare("#") != 0) {
14.       if (str.compare("YueBing") == 0) {
15.           myfile >> Gx >> Gy >> Gz >> Rx >> Ry >> Rz >> Sx >> Sy >> Sz >> type;
16.           YueBing* s = new YueBing(Gx, Gy, Gz, type);
17.           s->scaleX = Sx; s->scaleY = Sy; s->scaleZ = Sz;
18.           s->rotateX = Rx; s->rotateY = Ry; s->rotateZ = Rz;
19.           Shapes.push_back(s);
20.       }
21.       else if (str.compare("YueBingPi") == 0) {
22.           myfile >> Gx >> Gy >> Gz >> Rx >> Ry >> Rz >> Sx >> Sy >> Sz;
23.           YueBingPi* s = new YueBingPi(Gx, Gy, Gz);
24.           s->scaleX = Sx; s->scaleY = Sy; s->scaleZ = Sz;
25.           s->rotateX = Rx; s->rotateY = Ry; s->rotateZ = Rz;
26.           Shapes.push_back(s);
```

```
27.         }
28.         else if (str.compare("Sphere") == 0){
29.             myfile >> Gx >> Gy >> Gz >> Rx >> Ry >> Rz >> Sx >> Sy >> Sz >> type;
30.             Sphere* s = new Sphere(Gx, Gy, Gz, type);
31.             s->scaleX = Sx; s->scaleY = Sy; s->scaleZ = Sz;
32.             s->rotateX = Rx; s->rotateY = Ry; s->rotateZ = Rz;
33.             Shapes.push_back(s);
34.         }
35.         else if (str.compare("LiWuHePingMian") == 0){
36.             myfile >> Gx >> Gy >> Gz >> Rx >> Ry >> Rz >> Sx >> Sy >> Sz >> type;
37.             LiWuHePingMian* s = new LiWuHePingMian(Gx, Gy, Gz, type);
38.             s->scaleX = Sx; s->scaleY = Sy; s->scaleZ = Sz;
39.             s->rotateX = Rx; s->rotateY = Ry; s->rotateZ = Rz;
40.             Shapes.push_back(s);
41.         }
42.         else if (str.compare("LiWuHe") == 0){
43.             myfile >> Gx >> Gy >> Gz >> Rx >> Ry >> Rz >> Sx >> Sy >> Sz >> type;
44.             LiWuHe* s = new LiWuHe(Gx, Gy, Gz, type);
45.             s->scaleX = Sx; s->scaleY = Sy; s->scaleZ = Sz;
46.             s->rotateX = Rx; s->rotateY = Ry; s->rotateZ = Rz;
47.             Shapes.push_back(s);
48.         }
49.         myfile >> str;
50.     }
51.     myfile.close();
52.     break;
53.   }
```

3.3 模块功能

3.3.1 环境

实现在 drawEnvironment.cpp 文件中。

函数名	void InitList();
函数功能	初始化各物件的列表。
实现思路	调用各物件的对应 init 函数。

函数名	void initLightbulbList();
函数功能	初始化灯泡列表。
实现思路	利用 glNewList 函数新建灯泡列表，创建球体对象，设置所需属性并且绑定纹理。

代码:

```
void initLightbulbList() {
    //创建新的列表
    glNewList(bulb[0], GL_COMPILE);
    //创建图形对象
    GLUquadricObj* sphere = gluNewQuadric();
    //设置对应属性
    gluQuadricNormals(sphere, GLU_SMOOTH);
    gluQuadricDrawStyle(sphere, GLU_FILL);
    gluQuadricTexture(sphere, GL_TRUE);
    glPushMatrix();
    glEnable(GL_TEXTURE_2D);
    //绑定所需纹理
    glBindTexture(GL_TEXTURE_2D, 0);
    gluSphere(sphere, 0.2, 50, 50);
    glDisable(GL_TEXTURE_2D);
    glPopMatrix();
    glEndList();
}
```

函数名	GLint initWindowList() ;
函数功能	初始化窗户列表并返回在对应列表中的值。
实现思路	利用 glNewList 函数新建窗户列表，将窗户分为可滑动的窗玻璃和固定的打开的窗口两个部分，并且三扇窗映照出的夜景纹理不同，设置所需属性并且绑定纹理。

函数名	GLint initDoorList();
函数功能	初始化门列表并返回在对应列表中的值。
实现思路	利用 glNewList 函数新建门列表，将门分为门板、门外走廊、门框三个部分，其中门框由两侧和上方两个部分，一共有两扇门，设置所需属性并且绑定纹理。

函数名	GLint initWallList();
函数功能	初始化墙列表并返回在对应列表中的值。
实现思路	利用 glNewList 函数把基本的墙面元放入列表，分为天花板、地板和墙体三类，设置所需属性并且绑定纹理。

函数名	GLint initClosetList();
函数功能	初始化柜子列表并返回在对应列表中的值。
实现思路	利用 glNewList 函数把基本的柜面元放入列表，以不同的面来划分，设置所需材质属性并且绑定纹理。

函数名	void initDesk();
函数功能	初始化桌子列表。

实现思路	利用 glNewList 函数把基本的桌面元放入列表，设置所需材质属性并且绑定纹理。

函数名	unsigned char* LoadBitmapFile(char* filename, BITMAPINFOHEADER * bitmapInfoHeader);
函数功能	从数据文件中装载纹理数据。
实现思路	根据路径打开图片，遍历图片读取它的对应坐标的 RGB 数据，形成数组并返回。

函数名	void texload(int i, char* filename);
函数功能	读取指定路径的图片，并装载到对应纹理。
实现思路	调用 LoadBitmapFile()函数从数据文件中装载纹理数据，产生一个纹理标识符 ，指定过滤方式后，从纹理数据生成所需的纹理。

函数名	void initTexture();
函数功能	初始化纹理。
实现思路	利用 glGenTextures()生成纹理后，调用 texload()函数将指定的图片读取成对应编号的纹理，并且允许多重纹理映射。

函数名	void Change_Rust();
函数功能	改变公告板上形成的纹理。
实现思路	每次修改表示公告板纹理模式的全局变量 boardmode。

函数名	bool Change_Door_1();
函数功能	改变 1 号门的开闭状态。
实现思路	每次修改表示 1 号门的开闭状态的全局变量 r_door_on[0]并返回。

函数名	void Change_Door_2();
函数功能	改变 2 号门的开闭状态。
实现思路	每次修改表示 2 号门的开闭状态的全局变量 r_door_on[1]。

函数名	void Change_Window(int i);
函数功能	改变窗的开闭状态。
实现思路	输入需要开闭的窗的编号，每次修改表示对应编号的窗的开闭状态的全局变量 window_pos_on[i]。

函数名	void Texture_cube(int n, int i, int j, int k);
函数功能	绘制带有纹理的立方体，支持多重纹理映射。
实现思路	n 表示需要映射的纹理数量，i,j,k 分别代表了需要映射的纹理的编号，随后纹理会被分别绑定到对应坐标的面上，绘制出一个从(-1,-1,-1)到(1,1,1)的立方体。

函数名	void draw();

函数功能	绘制环境。
实现思路	调用各物体的绘制函数。

函数名	void drawoneWindow();
函数功能	绘制一扇窗户。
实现思路	调用窗元列表，通过 glTranslatef()和 glScalef()函数拼接成一扇窗户。其中每扇窗户利用 window_pos[i]变量表示窗玻璃当前坐标。

代码：

```
1.  //绘制一扇窗
2.  void drawoneWindow(int i) {
3.     glCallList(window[i + 1]);
4.     glPushMatrix();
5.     glTranslatef(0, 0.0043, window_pos[i]);//利用 window_pos[i]变量表示窗玻璃当前坐标
6.     glCallList(window[0]);
7.     glPopMatrix();
8.  }
```

函数名	void drawWindow();
函数功能	在场景中绘制窗户。
实现思路	通过 glTranslatef()调整窗户的坐标,并且调用 drawonewindow()函数绘制出窗框。其中每扇窗户利用 window_pos_on[i]变量判断处于开窗状态或者关窗状态，并且将玻璃对应的 window_pos[i]坐标进行相应的平移，每次重绘时增加一定平移量，从而达成动画效果。

代码：

```
1.  void drawWindow() {
2.     glPushMatrix();
3.     glTranslatef(8.0, 1.0, 2.0);
4.     glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
5.     drawoneWindow(0);//依次绘制三扇窗
6.     if (window_pos[0] <= 0.35 && window_pos_on[0]) window_pos[0] += 0.002;//如果窗户处于开启状
    态,且当前并未达到完全开启,那么将玻璃向右移动
7.     if (window_pos[0] >= -0.35 && !window_pos_on[0]) window_pos[0] -= 0.002;//如果窗户处于关闭状
    态,且当前并未达到完全闭合,那么将玻璃向左移动
8.     glTranslatef(0.0, 0.0, 2.0);
9.     glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
10.    drawoneWindow(1);
11.    if (window_pos[1] <= 0.35 && window_pos_on[1]) window_pos[1] += 0.002;
12.    if (window_pos[1] >= -0.35 && !window_pos_on[1])window_pos[1] -= 0.002;
13.    glTranslatef(0.0, 0.0, 2.0);
14.    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
15.    drawoneWindow(2);
16.    if (window_pos[2] <= 0.35 && window_pos_on[2]) window_pos[2] += 0.002;
17.    if (window_pos[2] >= -0.35 && !window_pos_on[2])window_pos[2] -= 0.002;
18.    glPopMatrix();
```

```
19. }
```

函数名	void drawGlass();
函数功能	在场景中绘制一块玻璃板，用于分隔生产区和外侧。
实现思路	玻璃板需要半透明效果，所以需要关闭 LIGHTENING 和打开 BLEND，通过 glTranslatef()和 glScalef()函数调整到合适的大小和位置后调用 glSolidCube()绘制即可。

代码：

```
1.  void drawGlass() {
2.    glPushMatrix();
3.    GLfloat mat_specular[] = { 1., 1., 1., 1.0 };          // 镜面反射颜色
4.    GLfloat mat_shininess[] = { 50.0 };                    // 镜面反射参数
5.    GLfloat lmodel_ambient[] = { 1., 1., 1., 1.0 };    // 散射颜色
6.    GLfloat lmodel_emmision[] = { 0.0, 0.0, 0.0, 1.0 };
7.
8.    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
9.    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
10.   glMaterialfv(GL_FRONT, GL_DIFFUSE, lmodel_ambient);
11.   glMaterialfv(GL_FRONT, GL_EMISSION, lmodel_emmision);
12.   glTranslatef(4.0, 1.0, 3.0);
13.   glEnable(GL_BLEND);//使创建的立方体可以产生半透明的效果
14.   glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
15.   glDepthMask(GL_FALSE);
16.   glDisable(GL_LIGHTING);//在绘制玻璃板的时候需要关闭灯光效果
17.   glColor4f(1.0, 1.0, 1.0, 0.2);
18.   glEnable(GL_NORMALIZE);//规格化向量
19.   glScalef(8.0, 2.0, 0.01);
20.   glutSolidCube(1.0);//绘制实心立方体
21.   glDisable(GL_BLEND);
22.   glEnable(GL_LIGHTING);//重新打开灯光
23.   glDepthMask(GL_TRUE);
24.   glPopMatrix();
25. }
```

函数名	void drawoneDesk();
函数功能	在场景中绘制一张桌子。
实现思路	调用桌面元列表，通过 glTranslatef()和 glScalef()函数拼接成一张桌子。

函数名	void drawDesk();
函数功能	在场景中绘制桌子。
实现思路	通过 glTranslatef()调整桌子绘制的坐标,并且调用 drawoneDesk()函数绘制出桌子。

函数名	void drawoneboard();

函数功能	绘制一块公告板。
实现思路	调用 Texture_cube()函数绘制带有所需纹理的立方体,并且通过 glTranslatef()和 glScalef()函数拼接成一块公告板。其中采用变量 broadmode 控制公告板上显示的内容:true 显示兔子造月饼的概念图，false 显示工厂生产的连续动图。broadnum 用于控制动图中显示的具体纹理，broadcounter 用于控制 broadnum 不同纹理之间显示的间隔时间。

函数名	void drawBoard();
函数功能	在场景中绘制公告板。
实现思路	通过 glTranslatef()调整桌子绘制的坐标,并且调用 drawoneboard()函数绘制出桌子。

函数名	void drawonecloset();
函数功能	绘制一个柜子。
实现思路	调用柜面元列表，通过 glTranslatef()和 glScalef()函数拼接成一个柜子。

函数名	void drawCloset();
函数功能	在场景中绘制柜子。
实现思路	通过 glTranslatef()调整柜子的坐标,并且调用 drawonecloset()函数绘制出柜子。

函数名	void drawLightbulb();
函数功能	在场景中绘制灯泡。
实现思路	需要绘制 4*4 的灯泡，故而使用 for 循环。通过 glTranslatef()调整灯泡的坐标,并且调用列表绘制出灯泡。考虑到光源数量的限制，绘制时通过修改 glMaterialfv()函数的参数，在对角线上采用聚光灯效果。

函数名	void drawOutsideDoor_1();
函数功能	绘制 1 号门框。
实现思路	通过 glTranslatef()调整门的坐标,并且调用元列表绘制门框以及门外走廊。

函数名	void drawOutsideDoor_2();
函数功能	绘制 2 号门框。
实现思路	通过 glTranslatef()调整门的坐标,并且调用元列表绘制门框以及门外走廊。

函数名	bool getRDoor_1() ;
函数功能	返回 1 号门的状态。
实现思路	返回 1 号门的开闭状态。

函数名	bool getRDoor_2() ;
函数功能	返回 2 号门的状态。
实现思路	返回 2 号门的开闭状态。

函数名	void drawDoor() ;

函数功能	在场景中绘制门。
实现思路	利用 glTranslatef() 调整坐标，调用 drawOutsideDoor_1() 和 drawOutsideDoor_2()函数绘制门框和门外走廊。r_door[i]则表示目前门板与墙之间的夹角。利用 glRotate()函数旋转门板到所需角度。r_door_on[i]表示门的开闭状态，如果门正在开启,增大门板的夹角，也就是增加 r_door[i]，从而达到连续的动画效果。

函数名	void setWall() ;
函数功能	设置墙相关材质系数。
实现思路	通过 glMaterialfv()函数设置即可。

函数名	void drawWall();
函数功能	绘制墙面。
实现思路	通过 glTranslatef()调整墙面的坐标,并且调用元列表绘制墙面、天花板和地板。

函数名	void initTile() ;
函数功能	初始化条带列表。
实现思路	根据飘动的角度不同，修改控制点 z 轴坐标，并且调用 gluNurbsSurface()函数绘制相应的 Nurbs 曲面到列表中。

函数名	void initNurbs();
函数功能	初始化扇叶列表。
实现思路	设置纹理和特殊效果之后，调用 gluNurbsSurface()函数绘制相应的 Nurbs 曲面到列表中。

函数名	void drawfans();
函数功能	在场景中绘制电扇。
实现思路	调用列表绘制出扇叶，并且通过 fan_r 变量控制扇叶的旋转角度，产生连贯的动画效果，并且绘制被风吹动的条带。

函数名	void mooncakeCollision();
函数功能	碰撞检测
实现思路	将 Shapes 复制按坐标排序（以尽可能提前终止子循环，减少时间复杂度）后，进行遍历检测，认为符合近距离要求的两个物体发生了碰撞，其后对两个物体的 type 进行识别，如果符合合成要求则在原 Shapes 中移除，并在碰撞位置添加合成后的新物体。

```
1.  // 检测并处理是否有月饼馅料与月饼皮、月饼与礼盒包装的碰撞
2.  void mooncakeCollision() {
3.      vector<Shape*> Shapes2;//复制一份到 Shapes2 用于 sort，不动原来的 Shapes
4.      Shapes2.assign ( Shapes.begin(), Shapes.end());
5.      vector<Shape*>::iterator Siter;
6.      vector<Shape*>::iterator subSiter;
7.      int i = 0;
8.      sort(Shapes2.begin(), Shapes2.end(), shapeCompare);
```

```
9.      bool collisionFlag = false;
10.   for (Siter = Shapes2.begin(); Siter + 1 != Shapes2.end();) {
11.       collisionFlag = false;
12.       if (abs((*Siter)->getGlobalX() - (*subSiter)->getGlobalX()) > 0.019) {
13.           break;
14.       }
15.       else if (abs((*Siter)->getGlobalZ() - (*subSiter)->getGlobalZ()) > 0.056
16.           || abs((*Siter)->getGlobalY() - (*subSiter)->getGlobalY()) > 0.016) {
17.           continue;
18.       }
19.       else {
20.           if (((*Siter)->RetType() == -
    2 && (*subSiter)->RetType() >= 0 && (*subSiter)->RetType() <= 2)
21.               || ((*subSiter)->RetType() == -
    2 && (*Siter)->RetType() >= 0 && (*Siter)->RetType() <= 2)) {
22.               // 馅料与饼皮相撞得月饼
23.               YueBing* mooncake = new YueBing((*Siter)->getGlobalX(), (*Siter)->getGlobalY(), (*Siter)->
    getGlobalZ(),
24.                   (*Siter)->RetType() == -2 ? (*subSiter)->RetType() + 10  : (*Siter)->RetType() + 10);
25.               Shapes.push_back(mooncake);//在原 Shapes 上进行改动
26.               collisionFlag = true;
27.               break;
28.           }
29.
30.           else if (((*Siter)->RetType() >= 10 && (*Siter)->RetType() <= 12 && ((*subSiter)->RetType() ==
    3 || (*subSiter)->RetType() == 4))
31.               || ((*subSiter)->RetType() >= 10 && (*subSiter)->RetType() <= 12 && ((*Siter)->RetType() ==
     3 || (*Siter)->RetType() == 4))) {
32.
33.               // 月饼与礼品盒皮相撞得礼盒
34.               LiWuHe* gift = new LiWuHe((*Siter)->getGlobalX(), (*Siter)->getGlobalY(), (*Siter)->getGlo
    balZ(),
35.                   (*Siter)->RetType() >= 10 ? (*subSiter)->RetType() + 20 : (*Siter)->RetType() + 20);

36.               Shapes.push_back(gift);
37.               collisionFlag = true;
38.               break;
39.           }
40.       }
41.   }
42.   if (collisionFlag) {    // 有相撞发生
43.       CurrentChooseShape = NULL;
44.       vector<Shape*>::iterator tempIter;
45.       for (tempIter = Shapes.begin(); tempIter != Shapes.end();) {
46.           if ((*tempIter) == (*Siter) || (*tempIter) == (*subSiter)) {
```

```
47.                 // cout << "*Siter:" << *Siter << ", * subSiter:" << *subSiter << ", tempIter:" << *tempIter << e
    ndl;
48.                 tempIter = Shapes.erase(tempIter);
49.             }
50.             else {
51.                 tempIter++;
52.             }
53.         }
54.         int bias = subSiter - Siter;
55.         Siter = Shapes2.erase(Siter);
56.         Shapes2.erase(Siter + bias - 1);
57.         break;
58.     }
59.     else {
60.         Siter++;
61.     }
62.  }
63. }
```

3.3.2 光照

函数名	void setLight();
函数功能	光源设置
实现思路	根据全局变量控制，打开或关闭 0-7 号光源，并对光源的颜色、衰减率进行设置。

函数名	draw 函数中的材质设置
函数功能	材质设置
实现思路	根据不同材质的特性，分别在 draw 函数中设置其漫反射、镜面反射的属性，包括颜色、反射率等。

3.3.3 物体

基本体素绘制主要实现在 Shape.h 和 Shape.cpp 中。

父类：

平移：

函数名	void Shape::transfer(double dx, double dy, double dz);
函数功能	改变物体的位置参数，从而将平移物体。
实现思路	将传入改变量 dx,dy,dz 分别加到 this->globalX,this->globalY,this->globalZ 上。

旋转：

函数名	void Shape::rotate(double dx, double dy, double dz);

函数功能	改变物体的旋转参数，从而将旋转物体。
实现思路	将传入改变量 dx,dy,dz 分别加到 this->rotateX,this->rotateY,this->rotateZ 上。

缩放：

函数名	void Shape::scaling(float scaleX, float scaleY, float scaleZ)
函数功能	改变物体的缩放参数，从而将物体沿 x,y,z 轴缩放。
实现思路	将传入改变量 dx,dy,dz 分别与 this->rotateX,this->rotateY,this->rotateZ 相乘。

子类：在 Shape 父类上继承实现立方体、圆柱等体素，主要实现了 draw 函数进行绘制。

六棱柱体：

函数名	void Prism::Draw()
函数功能	绘制带纹理的六棱柱体。
实现思路	首先进行纹理和光照相关参数的设置； 位置姿态变换：旋转、缩放、平移； 具体绘制：先定义六棱柱 12 个顶点的坐标数组 vtx[12][3]，再使用 glBegin(GL_POLYGON); 分别画出六棱柱上下底面及 6 个侧面。

代码：

在平移旋转缩放部分，由于执行顺序是从后到前，所有函数的顺序是 glTranslatef, glScalef, glRotatef.

```
1.  void Prism::Draw()
2.  {
3.      //光照和纹理处理
4.      glEnable(GL_TEXTURE_2D);
5.      glBindTexture(GL_TEXTURE_2D, texture[this->Texture]);
6.      GLfloat mat_specular[] = { 1., 1., 1., 1.0 };          // 镜面反射颜色
7.      GLfloat mat_shininess[] = { 50.0 };                    // 镜面反射参数
8.      GLfloat lmodel_ambient[] = { 1., 1., 1., 1.0 };    // 散射颜色
9.      GLfloat lmodel_emmision[] = { 0.0, 0.0, 0.0, 1.0 };
10.     glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
11.     glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
12.     glMaterialfv(GL_FRONT, GL_DIFFUSE, lmodel_ambient);
13.     glMaterialfv(GL_FRONT, GL_EMISSION, lmodel_emmision);
14.     //平移旋转缩放
15.     glPushMatrix();
16.     glEnable(GL_NORMALIZE);
17.     glTranslatef(this->globalX, this->globalY, this->globalZ);
18.     glScalef(0.2, 0.2, 0.2);//这是为了默认画出来的 shape 大小不要太夸张
19.     glScalef(this->scaleX, this->scaleY, this->scaleZ);
20.     glRotatef(this->rotateX, 1, 0, 0);
21.     glRotatef(this->rotateY, 0, 1, 0);
22.     glRotatef(this->rotateZ, 0, 0, 1);
23.     //定义棱柱顶点坐标
24.     static GLfloat vtx[12][3] =
```

```
25.     {
26.       //0-5 下层，
27.       {-0.5f,0.0f,0.0f},//0
28.       {-0.25f,0.0f,static_cast<GLfloat>(sqrt(3.0f) / 4)},//1
29.       {-0.25f,0.0f,-static_cast<GLfloat>(sqrt(3.0f) / 4)},//2
30.       {0.25f,0.0f,static_cast<GLfloat>(sqrt(3.0f) / 4)},//3
31.       {0.25f,0.0f,-static_cast<GLfloat>(sqrt(3.0f) / 4)},//4
32.       {0.5f,0.0f,0.0f},//5
33.       {-0.5f,0.5f,0.0f},//6
34.       {-0.25f,0.5f,static_cast<GLfloat>(sqrt(3.0f) / 4)},//7
35.       {-0.25f,0.5f,-static_cast<GLfloat>(sqrt(3.0f) / 4)},//8
36.       {0.25f,0.5f,static_cast<GLfloat>(sqrt(3.0f) / 4)},//9
37.       {0.25f,0.5f,-static_cast<GLfloat>(sqrt(3.0f) / 4)},//10
38.       {0.5f,0.5f,0.0f},//11
39.     };
40.     //六棱柱下底面（其实想要多少面的棱柱都行，算数然后画就好了）
41.     glBegin(GL_POLYGON);
42.     glNormal3f(0.0, 0.0, 1.0);
43.     glTexCoord2i(0.75, 1); glVertex3fv(vtx[2]);
44.     glTexCoord2i(1, 0.5); glVertex3fv(vtx[0]);
45.     glTexCoord2i(0.75, 0); glVertex3fv(vtx[1]);
46.     glTexCoord2i(0.25, 0); glVertex3fv(vtx[3]);
47.     glTexCoord2i(0, 0.5); glVertex3fv(vtx[5]);
48.     glTexCoord2i(0.25, 1); glVertex3fv(vtx[4]);
49.     glEnd();
50.
51.     //中间六个面
52.     glBegin(GL_QUADS);
53.     glNormal3f(0.0, 0.0, 1.0);
54.     glTexCoord2i(1, 1); glVertex3fv(vtx[10]);
55.     glTexCoord2i(1, 0); glVertex3fv(vtx[4]);
56.     glTexCoord2i(0, 0); glVertex3fv(vtx[8]);
57.     glTexCoord2i(0, 1); glVertex3fv(vtx[2]);
58.
59.     glNormal3f(0.0, 0.0, 1.0);
60.     glTexCoord2i(1, 1); glVertex3fv(vtx[8]);
61.     glTexCoord2i(1, 0); glVertex3fv(vtx[2]);
62.     glTexCoord2i(0, 0); glVertex3fv(vtx[0]);
63.     glTexCoord2i(0, 1); glVertex3fv(vtx[6]);
64.
65.     glNormal3f(0.0, 0.0, 1.0);
66.     glTexCoord2i(1, 1); glVertex3fv(vtx[11]);
67.     glTexCoord2i(1, 0); glVertex3fv(vtx[5]);
68.     glTexCoord2i(0, 0); glVertex3fv(vtx[4]);
69.     glTexCoord2i(0, 1); glVertex3fv(vtx[10]);
```

```
70.
71.     glNormal3f(0.0, 0.0, 1.0);
72.     glTexCoord2i(1, 1); glVertex3fv(vtx[6]);
73.     glTexCoord2i(1, 0); glVertex3fv(vtx[0]);
74.     glTexCoord2i(0, 0); glVertex3fv(vtx[1]);
75.     glTexCoord2i(0, 1); glVertex3fv(vtx[7]);
76.
77.     glNormal3f(0.0, 0.0, 1.0);
78.     glTexCoord2f(1.0, 1.0); glVertex3fv(vtx[7]);
79.     glTexCoord2f(1.0, 0.0); glVertex3fv(vtx[1]);
80.     glTexCoord2f(0.0, 0.0); glVertex3fv(vtx[3]);
81.     glTexCoord2f(0.0, 1.0); glVertex3fv(vtx[9]);
82.
83.     glNormal3f(0.0, 0.0, 1.0);
84.     glTexCoord2i(1, 1); glVertex3fv(vtx[9]);
85.     glTexCoord2i(1, 0); glVertex3fv(vtx[3]);
86.     glTexCoord2i(0, 0); glVertex3fv(vtx[5]);
87.     glTexCoord2i(0, 1); glVertex3fv(vtx[11]);
88.     glEnd();
89.
90.     //六棱柱上底面
91.     glBegin(GL_POLYGON);
92.     glNormal3f(0.0, 0.0, 1.0);
93.     glTexCoord2i(0.75, 1); glVertex3fv(vtx[8]);
94.     glTexCoord2i(1, 0.5); glVertex3fv(vtx[6]);
95.     glTexCoord2i(0.75, 0); glVertex3fv(vtx[7]);
96.     glTexCoord2i(0.25, 0); glVertex3fv(vtx[9]);
97.     glTexCoord2i(0, 0.5); glVertex3fv(vtx[11]);
98.     glTexCoord2i(0.25, 1); glVertex3fv(vtx[10]);
99.     glEnd();
100.
101.       glPopMatrix();
102.       glDisable(GL_TEXTURE_2D);
103.   }
```

六棱台：

函数名	void Trustum::Draw()
函数功能	绘制带纹理的正六棱台。
实现思路	首先进行纹理和光照相关参数的设置； 位置姿态变换：旋转、缩放、平移； 具体绘制：先定义六台 12 个顶点的坐标数组 vtx[12][3]，再使用 glBegin(GL_POLYGON); 分别画出六棱台上下底面及 6 个侧面。

立方体：

函数名	void Cube::Draw()

函数功能	绘制带纹理的立方体。
实现思路	首先进行纹理和光照相关参数的设置； 位置姿态变换：旋转、缩放、平移； 具体绘制：调用 void Texture_cube(int n, int i, int j, int k)函数，n 表示采用纹理的数量,i,j,k 是纹理的序号,具体绘制方式和六棱柱类似。

圆锥：

函数名	void Cone::Draw()
函数功能	圆锥的 draw 函数。
实现思路	首先进行纹理和光照相关参数的设置； 位置姿态变换：旋转、缩放、平移； 具体绘制：调用 void Cone::Texture_Cone(int i)函数，i 表示是纹理的序号。

函数名	void Cone::Texture_Cone(int i)
函数功能	具体绘制带纹理的圆锥。
实现思路	首先进行纹理和光照相关参数的设置，这里使用了 glEnable(GL_TEXTURE_GEN_S);和 glEnable(GL_TEXTURE_GEN_T);来自动生成纹理坐标； 具体绘制：调用 void glutSolidCone(GLdouble radius, GLdouble height, GLint slices,GLint stacks);函数，参数为圆锥底面半径、圆锥的高，后两个参数为圆锥经线数量和纬线数量。

圆台：

函数名	void ConeCylinder::Texture_Cone(int i)
函数功能	圆台的 draw 函数。
实现思路	首先进行纹理和光照相关参数的设置； 位置姿态变换：旋转、缩放、平移； 具体绘制：调用 void ConeCylinder::Texture_ConeCylinder(int i);函数

函数名	void ConeCylinder::Texture_ConeCylinder(int i);
函数功能	具体绘制带纹理的圆台，包括上下底面。
实现思路	首先进行纹理和光照相关参数的设置； 具体绘制：调用 void gluCylinder (GLUquadric *qobj, GLdouble baseRadius, GLdouble topRadius, GLdouble height, GLint slices, GLint stacks);绘制一个沿 z 轴的圆柱体，底面位于 z=0 平面，顶面位于 z=height 的平面。baseRadius 是 z=0 半径，topRadius 是 z=height 半径。 但上述绘制的只是圆台的侧面，上下底面需另外绘制，调用 void ConeCylinder::Texture_Circle(int i, float r)函数进行绘制。

代码：

```
1.  GLUquadricObj* left1 = gluNewQuadric();
2.  gluQuadricDrawStyle(left1, GLU_FILL);
3.  gluQuadricTexture(left1, GL_TRUE);//启用纹理，可以绑定不同的纹理进行贴图
4.  gluCylinder(left1, 1.0, 0.5, 1.0, 15, 5);
5.  Texture_Circle(i, 1);
6.  glTranslatef(0, 0, 1);
```

```
7.  Texture_Circle(i, 0.5);
```

函数名	void ConeCylinder::Texture_Circle(int i, float r)
函数功能	绘制圆台的上、下底面。
实现思路	首先进行纹理和光照相关参数的设置； 具体绘制：调用 void gluDisk (GLUquadric *qobj, GLdouble innerRadius, GLdouble , GLint slices, GLint loops);绘制一个位于 z=0 处的圆盘，innerRadius 是内半径，outerRadius 是外半径，loops 是几个同心圆。这里设置 innerRadius=0，得到的就是圆形。

圆柱：和圆台的绘制方法一模一样，只需 gluCylinder 函数的上下底面设置成一样即可，故不赘述。

月饼皮：继承圆柱类，只需在构造函数中规定纹理和大小即可，故不赘述。

月饼：继承圆柱类，只需在构造函数中规定纹理和大小即可。需要注意的是，由于月饼有三种口味，构造函数可以用 YueBing(float globalX, float globalY, float globalZ)只规定坐标，将纹理设置为默认种类；也可以增加一个 Type 变量来在创建时控制月饼的种类。

函数名	void YueBing::SetType(int t);
函数功能	设置月饼的种类。
实现思路	根据输入的参数修改月饼中的 Type 变量。

球体（月饼馅料）：由于月饼有三种口味，构造函数可以用 Sphere(float globalX, float globalY, float globalZ)只规定坐标，将纹理设置为默认种类；也可以增加一个 Type 变量来在创建时控制馅料的种类。

函数名	void Sphere::Draw();
函数功能	月饼馅料的 draw 函数。
实现思路	首先进行纹理和光照相关参数的设置； 位置姿态变换：旋转、缩放、平移； 具体绘制：调用 void Sphere::Texture_Sphere(int i);函数，i 表示是纹理的序号。

函数名	void Sphere::Texture_Sphere(int i);
函数功能	具体绘制带纹理的球体。
实现思路	首先进行纹理和光照相关参数的设置； 具体绘制：创建一个新的 GLUquadricObj 对象指针，调用 void gluSphere (GLUquadric *qobj, GLdouble Radius, GLint slices, GLint stacks);绘制一个球体，Radius 是球的半径。

代码：

```
1.  glPushMatrix();
2.  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
3.  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
4.
```

```
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_TEXTURE_2D);

    GLUquadricObj* sphere = gluNewQuadric();
    gluQuadricNormals(sphere, GLU_SMOOTH);
    gluQuadricDrawStyle(sphere, GLU_FILL);
    gluQuadricTexture(sphere, GL_TRUE); //启用纹理，可以绑定不同的纹理进行贴图
    gluSphere(sphere, 1.0, 50, 50);
    glPopMatrix();
    glDisable(GL_TEXTURE_2D);
}
```

礼物盒平面图：继承了 Cube 类。

函数名	void LiWuHePingMian::Draw();
函数功能	礼物盒平面的 draw 函数。
实现思路	首先进行纹理和光照相关参数的设置； 位置姿态变换：旋转、缩放、平移； 具体绘制：通过平移缩放，调用 void Texture_Cube(int n,int i,int j,int k);函数绘制出多个立方体进行拼接。

代码：

```
        Texture_cube(1, this->Texture, 0, 0);
    glTranslatef(1.5, 0, 0);
    glScalef(0.5, 1, 1);
    Texture_cube(1, this->Texture, 0, 0);
    glTranslatef(-6, 0, 0);
    Texture_cube(1, this->Texture, 0, 0);
    glTranslatef(3, 0, 1.5);
    glScalef(2, 1, 0.5);
    Texture_cube(1, this->Texture, 0, 0);
    glTranslatef(0, 0, -6);
    Texture_cube(1, this->Texture, 0, 0);
    glScalef(1, 1, 2);
    glTranslatef(3, 0, 1.5);
    Texture_cube(1, this->Texture, 0, 0);
```

礼物盒：继承了 Cube 类。

函数名	void LiWuHe::Draw();
函数功能	礼物盒的 draw 函数。
实现思路	首先进行纹理和光照相关参数的设置； 位置姿态变换：旋转、缩放、平移； 具体绘制：通过平移缩放，调用 void Texture_LiWuHe(int i); 和 void Texture_Cube(int n,int i,int j,int k);函数绘制出多个立方体进行拼接。

函数名	void LiWuHe::Texture_LiWuHe(int i);
函数功能	具体绘制带纹理的立方体。
实现思路	具体绘制：与 void Texture_Cube(int n,int i,int j,int k);类似，但具体进行纹理和坐标的绑定时为满足需求进行了一定调整。

代码：

```
1.      glBegin(GL_QUADS);
2.      glNormal3f(0.0, 0.0, 1.0);
3.      glTexCoord2i(1, 1); glVertex3i(-1, 1, 1);
4.      glTexCoord2i(1, 0); glVertex3i(-1, -1, 1);
5.      glTexCoord2i(0, 0); glVertex3i(1, -1, 1);
6.      glTexCoord2i(0, 1); glVertex3i(1, 1, 1);
7.
8.      ......    //类似不多赘述
9.
10.     glNormal3f(0.0, -1.0, 0.0);
11.     glTexCoord2i(1, 1); glVertex3i(-1, -1, 1);
12.     glTexCoord2i(1, 0); glVertex3i(-1, -1, -1);
13.     glTexCoord2i(0, 0); glVertex3i(1, -1, -1);
14.     glTexCoord2i(0, 1); glVertex3i(1, -1, 1);
15.
16.     glEnd();
```

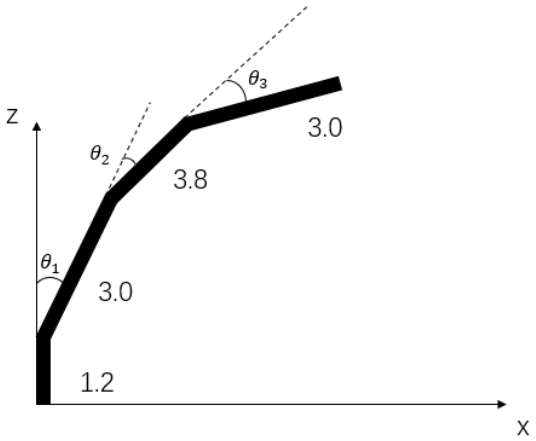
3.3.4 机械臂

实现在 Robot.h 和 Robot.cpp 文件中。

机械臂分为一个父类 Robot 和两个子类 Robot_1 和 Robot_2。其中父类比较简单，定义了机械臂在世界坐标系中的位置和绑定的 Shape，以及绘制函数，由于绘制函数由子类继承实现，不同子类实现过程不同，因此在两个子类中详细介绍。

Robot_2 子类实现除机械爪之外机械臂的绘制与各种功能。机械臂一共有四个自由度：第一个自由度决定机械臂整体的朝向，也就是以 y 轴为旋转轴的旋转角度，第二个自由度为第一个关节相对于原始位置的旋转角度，第三个自由度为第二个关节相对于原始位置的旋转角度，第四个自由度为第三个关节相对于原始位置的旋转角度。示意图可以在 x-z 坐标系中说明，如下所示：

机械臂 2 由关节和杆构成，而关节和杆实际上是由圆柱体组成的，因此机械臂全部是由



圆柱体堆叠组成的，事实上这种最简单的圆柱体也能最终形成效果不错的机械臂。

绘制杆的函数说明如下所示：

函数名	DrawRod(GLdouble baseR, GLdouble topR, GLdouble h)
函数功能	绘制机械臂的杆，baseR 为杆下底面的半径，topR 为杆上底面的半径，h 为圆柱的高
实现思路	分别绘制圆柱面的上下圆和侧面，并分别贴上相同的金属材质

绘制关节函数：

函数名	DrawJoint()
函数功能	绘制关节
实现思路	由两个圆柱体垂直组成，两个圆柱半径不同，有相交部分，上面的圆柱为下一个关节提供滑动的空间

绘制机械臂函数：

函数名	Draw()
函数功能	综合关节和杆，可以绘制出机械臂的全体：
实现思路	首先设置灯光效果，然后设置抓取机械爪的动画效果，然后绘制机械臂主题

由于该绘制函数涉及到比较多的几何变换关系，因此需要很多 push 和 pop 过程，因此将绘制过程按照四个自由度分开，用不同的缩进来表示不同的自由度层面。值得说明的是，设置机械臂所有自由度的初始状态为从原点指向 y 轴正方向，而后几个自由度实现方式为将该自由度以及该自由度之上的所有自由度向 y 轴负方向移动到原点，然后旋转自由度的角度，然后再移动回原来的位置，具体代码如下所示：

```
1.  /draw
2.  glPushMatrix();
3.  glTranslatef(PositionX, PositionY - 0.099, PositionZ);
4.  glEnable(GL_NORMALIZE); glScalef(1.0/6.0 * ZoomIndex, 1.0 / 6.0 * ZoomIndex, 1.0 / 6.0 * ZoomIndex)
    ;
5.      glPushMatrix();
6.    glRotatef(rotate1, 0, 1, 0);
7.        glPushMatrix();
8.      glTranslatef(0, 1.2, 0);
9.        glRotatef(rotate2, 1, 0, 0);
10.     glTranslatef(0, -1.2, 0);
11.           glPushMatrix();
12.       glTranslatef(0, 4.2, 0);
13.           glRotatef(rotate3, 1, 0, 0);
```

```
14.         glTranslatef(0, -4.2, 0);
15.             glPushMatrix();
16.             glTranslatef(0, 8, 0);
17.             glRotatef(rotate4, 1, 0, 0);
18.             glTranslatef(0, -8, 0);
19.             //机械臂 3
20.             glPushMatrix();
21.             glTranslatef(0, 8, 0);
22.             glRotatef(90, 0, 0, 1);
23.             glRotatef(90, 0, 1, 0);
24.             DrawRod(0.2, 0.1, 2);
25.             glPopMatrix();
26.             glPushMatrix();
27.             glTranslatef(0, 10, 0);
28.             glRotatef(180, 1, 0, 0);
29.             glScalef(0.25, 0.25, 0.25);
30.             Draw_Robot();
31.             glPopMatrix();
32.             glPopMatrix();
33.         //关节 3
34.         glPushMatrix();
35.         //glRotatef(-90, 1, 0, 0);
36.         glTranslatef(0, 7.4, 0);
37.         DrawJoint();
38.         glPopMatrix();
39.         //机械臂 2
40.         glPushMatrix();
41.         glTranslatef(0, 4.6, 0);
42.         glRotatef(90, 0, 0, 1);
43.         glRotatef(90, 0, 1, 0);
44.         DrawRod(0.2, 0.2, 2.8);
45.         glPopMatrix();
46.         glPopMatrix();
47.     //关节 2
48.     glPushMatrix();
49.     glTranslatef(0, 3.6, 0);
50.     DrawJoint();
51.     glPopMatrix();
52.     //机械臂 1
53.     glPushMatrix();
54.     glTranslatef(0, 1.6, 0);
55.     glRotatef(90, 0, 0, 1);
56.     glRotatef(90, 0, 1, 0);
57.     DrawRod(0.2, 0.2, 2);
58.     glPopMatrix();
```

```
59.         glPopMatrix();
60.     //关节 1
61.     glPushMatrix();
62.     glTranslatef(0, 0.6, 0);
63.     DrawJoint();
64.     glPopMatrix();
65.     glPopMatrix();
66.     //底座
67.     glPushMatrix();
68.     glRotatef(90, 0, 0, 1);
69.     glRotatef(90, 0, 1, 0);
70.     DrawRod(0.7, 0.5, 0.6);
```

上面提到，绘制机械臂函数的第二个步骤为设置机械爪抓取动画，该动画实现如下：

函数名	HandleRotate()
函数功能	设置机械爪抓取动画的函数
实现思路	如果 IsBind 为 true，说明当前机械臂与某一 shape 绑定，则设置动画效果。具体来说，动画流程由一个逐帧增加的变量 timeflag 决定：timeflag 从 0 到 1，机械臂从初始位置移动到需要抓取物体的位置；timeflag 从 1 到 2，机械爪收缩来抓取物体；timeflag 从 2 到 3，机械臂从抓取物体位置移动到放置物体位置，并在过程中改变绑定的 shape 的坐标（通过 SetBindPosition()函数完成）；timeflag 从 3 到 3.5，机械爪松开放下物体；timeflag 从 3.5 到 4.5，机械臂从放置物体位置移动回初始位置；timeflag 大于 4.5，动画过程结束，与 shape 解开绑定，将 timeflag 置为-1。

上面提到了需要在机械臂移动过程中设置 shape 的坐标，使它和机械爪同步移动，实现如下：

函数名	SetBindPosition()
函数功能	设置绑定 shape 的坐标
实现思路	实现思路如之前所画出的各自由度示意图所示，从第二个自由度推到第四个自由度，求出 shape 相对于机械臂原点的 x 和 y 坐标，再根据第一个自由度和上面得到的 x 坐标求出最终的 x 和 z 坐标。利用到了一些平面内的三角函数，而之所以能将空间中机械臂的移动效果转换到平面内，是由好的初始状态设置决定的。

3.3.5 传送带

主要实现在 conveyor.h, conveyor.cpp 中，在 main.cpp 中有相应的接口。
传送带实现相关函数：

函数名	conveyor::conveyor(float PositionX, float PositionY, float PositionZ, float MotionX, float MotionZ)
函数功能	传送带类的构造函数，确定传送带的位置以及其上物体的初速度。
实现思路	对 this->PositionX 等进行赋值。

函数名	void conveyor::rotate(double dx, double dy, double dz)
函数功能	传送带放置方向：绕 x,y,z 轴旋转的角度。
实现思路	将 this->ConrotateX 等加上传入的对应数值。

函数名	void conveyor::DrawConveyor2()
函数功能	传送带绘制函数。
实现思路	1）传送带两端的轮子是圆柱，参照圆柱的实现方法并贴上纹理即可； 2）传送带上下的带是两个立方体。 3）通过更换纹理图片表现传送带的运动。

传送带运动的实现：

1）在 conveyor 类中有变量”count”,count 等于 0-4 时会在 5 张相邻纹理间依次切换，视觉上形成传送带在运动的效果。

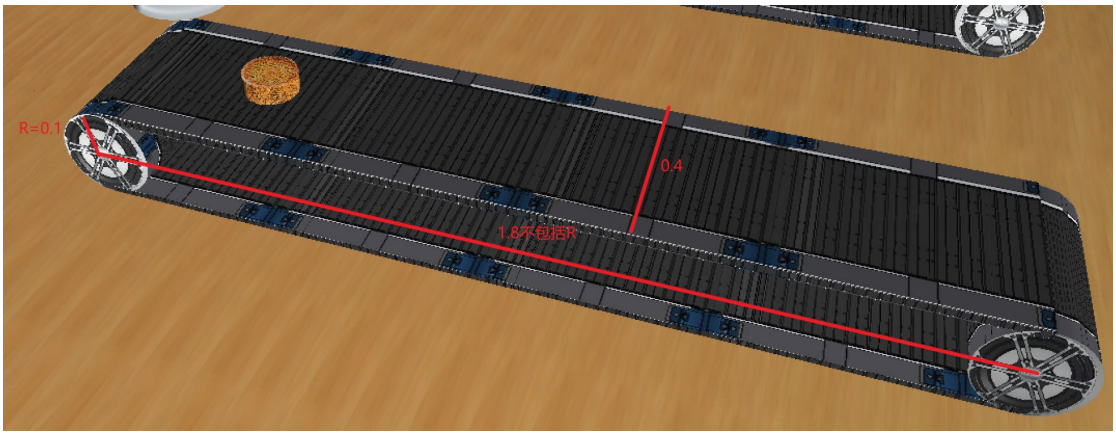
2）在 main.cpp 中的 key 函数中，r,t,h,j 用于控制不同传送带运动。按下按键后，会改变 count 值，并检查其上是否有物品，有则一起运动。

```
case 't':
  {
      Conveyors[2]->count = (Conveyors[2]->count + 1) % 5;
    AddMotionToShapes(Conveyors[2]);
      break;
  }
```

main.cpp 中的 void AddMotionToShapes(conveyor* Conv)函数将遍历 Shapes 容器，并确定每个 shape 是否要随该传送带运动。

```
void AddMotionToShapes(conveyor* Conv) {
  vector<Shape*>::iterator Siter;
    for (Siter = Shapes.begin(); Siter != Shapes.end(); Siter++) {
    Conv->AddMotion((*Siter));
    }
}
```

传送带尺寸示意图：



传送带和物体相关函数、接口：

函数名	bool conveyor::ifOntheConveyor(Shape* shape)
函数功能	判断某个 Shape 是否在传送带上。
实现思路	1）首先通过 shape->getGlobalX()、getGlobalY()、getGlobalZ()获取这个 Shape 的坐标；然后与传送带的世界坐标 this->PositionX、this->PositionY、this->PositionZ 比较，从而判断该 Shape 是否在此传送带上。 2）判断标准：x 方向满足”shapeX >= this->PositionX - 0.72 && shapeX<= this->PositionX + 0.91”: 由于程序中所使用的传送带是平行于 x 轴的，故判断条件只考虑传送带平行于 x 轴的时候。当传送带沿 x 轴负方向向正方向运动时，”+0.91”为起始方向，设置得较大则可以使得传送带边缘的物品也能判断为在传送带上从而被运走，”-0.72”为终点方向，设置得较小则不允许物体运动得太过，以免影响机械臂的判断； y 方向满足”shapeY > 0.16 && shapeY<0.3”, 实际上传送带上表面是 0.2，这里设置得较宽； z 方向满足”shapeZ>this->PositionZ - 0.2 && shapeZ < this->PositionZ + 0.2”，因为传送带宽 0.4，即物体不能超出传送带面。

函数名	void conveyor::AddMotion(Shape* shape)
函数功能	给在传送带上的 Shape 添加和传送带相适应的速度。
参数	#define K 0.0174532925　　//K=3.1415926/180,用于角度制和弧度制的转换，因为 c++的 sin cos 函数用的是弧度制
实现思路	1）如果 ifOntheConveyor 函数返回值为真，则设置 this->MotionX 和 this->MotionZ（没有 this->MotionY 是因为默认传送带只沿水平方向传送）。 2）通过 shape->transfer 函数将 this->MotionX 和 this->MotionZ 的值赋给物体。

代码：0.1 表示物体的速度，this->MotionX 和 this->MotionZ 是物体速度在 x 和 z 轴方向上的分量，乘上传送带绕 y 轴旋转角度 this->ConrotateY 的正弦或余弦即可。

```
1.  void conveyor::AddMotion(Shape* shape)//若 shape 在传送带上就运动
2.  {
3.      if (ifOntheConveyor(shape))
4.      {
5.          this->MotionX = -0.1 * cos(this->ConrotateY * K);
6.          this->MotionZ = 0.1 * sin(this->ConrotateY * K);
7.          shape->transfer(this->MotionX, 0, this->MotionZ);
8.      }
9.      return ;
10. }
```

最后，传送带和开门功能配合，当检测到最后一个传送带（最靠近门的传送带）的末端有月饼礼盒，且门为打开状态时，礼盒“消失”，表示合成好的礼盒被送出工厂。

代码：flag 为 true 即门被打开，此时进行判断，当传送带 5 末端（通过 shape 和传送带的 x,y,z 坐标进行判断）上有礼盒（type 为 23 或 24），则消除它。

```
1.  case 'f':
2.      {
3.          bool flag=Change_Door_1();
4.          //如果传送带 5 的末端有礼盒，则消失，表示礼盒被送出工厂
```

```
        if (flag) {
            vector<Shape*>::iterator Siter;
            for (Siter = Shapes.begin(); Siter != Shapes.end(); ) {
                if (((*Siter)->getGlobalX() <= 0.94)
                    && (abs((*Siter)->getGlobalY() - 0.2) < 0.05)
                    && (abs((*Siter)->getGlobalZ() - 6) <= 0.2)
                    && ((*Siter)->RetType() == 23 || (*Siter)->RetType() == 24))
                {
                    cout << "siterX:"<<(*Siter)->getGlobalX() << endl;
                    Siter = Shapes.erase(Siter);
                }
                else Siter++;
            }
        }
        break;
    }
```

3.3.6 相机漫游

程序整体设计相机漫游采用类 FPS 游戏的设计

变量名	变量作用	值域
screenrate_y	视线方向与 x 轴平面所成的线面角	[-180, 180]
screenrate_x	视线投影到 x 轴平面与 x 轴的夹角	[-89,89]

使用上述变量即可唯一确定视线方向的欧拉角，用来计算视点的位置与方向。
同时从用户的角度看，拖动鼠标会修改实现的方向，即鼠标的响应函数需要去更新上述两个变量的值,当鼠标相较于屏幕中心移动**(offsetx,offsety)**对应的(screenrate_x, screenrate_y)需要增加一个(offsetx/2/centerpoint_x*PI,offsety/2/centerpoint_y *PI) 的增量，同时需要对(screenrate_x, screenrate_y) 进行值域的校验，如果超出值域范围，将其固定在边界上。 结束流程后将鼠标重置回屏幕中心，等待下一次消息循环。

```
void mouse_move(int mx, int my)
{
    float offsetx = mx - OriX;
    float offsety = my - OriY;
    POINT p;
    screenrate_x = screenrate_x + (offsetx / 2 / centerpoint_x * PI);
    screenrate_y = screenrate_y + (-offsety/ 2 / centerpoint_y * PI);
    screenrate_y = screenrate_y < -cons ? -cons : (screenrate_y > cons ? cons : screenrate_y);
    SetCursorPos(wPosW + centerpoint_x + 8, wPosH + centerpoint_y + 31);
    OriX = centerpoint_x;
    OriY = centerpoint_y;
}
```

同时使用方向键控制相机位置时，相机不再沿着 x 轴 y 轴 z 轴方向移动，当使用前后方向键

前后移动时，相机会沿着视线方向做匀速运动，当使用左右方向键左右移动时，相机会沿着方向在 y=0 平面内投影的法线方向移动。

```
case 'a': {//todo, hint: eye[] and center[] are the keys to solve the problems
    eye[0] += step * cos(screenrate_x - PI / 2);
    eye[2] += step * sin(screenrate_x - PI / 2);
    break;
}
case 'd': {//todo
    eye[0] -= step * cos(screenrate_x - PI / 2);
    eye[2] -= step * sin(screenrate_x - PI / 2);
    break;
}
case 'w': {//todo
    //eye[2] -= 0.1;
    eye[1] += step * sin(screenrate_y);
    eye[0] += step * cos(screenrate_y) * cos(screenrate_x);
    eye[2] += step * cos(screenrate_y) * sin(screenrate_x);
    break;
}
case 's': {//todo
    eye[1] -= step * sin(screenrate_y);
    eye[0] -= step * cos(screenrate_y) * cos(screenrate_x);
    eye[2] -= step * cos(screenrate_y) * sin(screenrate_x);
    break;
}
case 'z': {//todo
    eye[1] += 0.1;

    break;
}
case 'c': {//todo
    if (eye[1] >= -0.35)
    {
        eye[1] -= 0.05;
    }
    break;
}
}
```

3.3.7 点击拾取

姓名栈设计，定义枚举变量，枚举选中的类型

```
typedef enum {
    __robot = 100, __shape, __conveyor, NONE
```

```
3.  }TYPE;
```

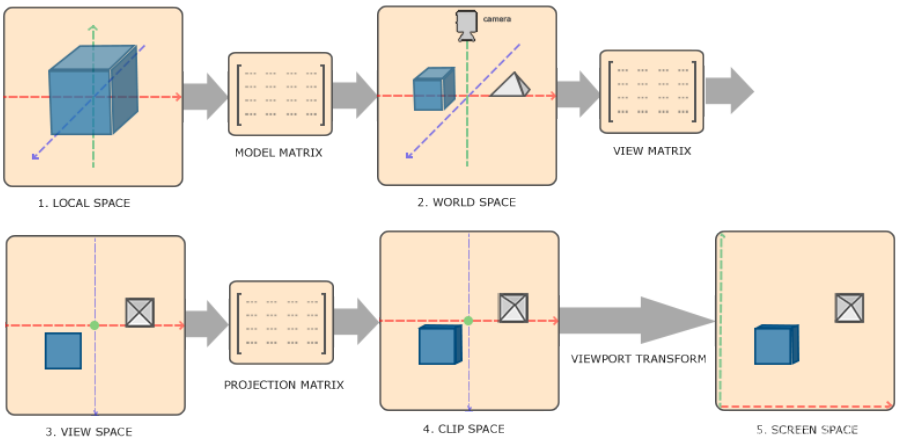
s_type
s_id
s_type
s_id
1. ....

当 Picking 模式获取到一个物块，其姓名栈的结构如上图所示，从姓名栈中取出最接近点击位置的二元组 (s_type,s_id) 即可唯一确定选中的物块。

同时，进入选择模式后使用正视图渲染画面之后使用选择框拾取物块的准确度更高，用户的体验也更好。

```
1.  glGetIntegerv(GL_VIEWPORT, viewport);//返回当前视口的数值，并存入 viewport 数组中
2.  gluPickMatrix(cursorX, viewport[3] - cursorY, 0.5, 0.5, viewport); //建立拾取矩阵，前两个参数为将窗口坐标 cursor 转化为 OpenGL 坐标，第三、四个参数是选择框的大小，最后一个就是视口的位置和大小
3.  glOrtho(-10, 10, -10, 10, -10, 10);
```

3.3.8 点击放置

函数名	glm::vec3 getViewPos(int x, int y, GLfloat* pro, GLfloat* view)
函数功能	实现屏幕坐标系下的坐标，到三维世界坐标系下的转换
实现原理	Opengl 中坐标系得变换  执行上述变换的逆过程即可得到屏幕坐标系下的鼠标坐标所对应的世界坐标。 正常计算裁剪坐标的公式为: $V_{clip} = M_{projection} * M_{view} * M_{model} * V_{local}$

	那么由裁剪坐标反向获取世界坐标的公式就是 $$M_{model} * V_{local} = M_{view}^{-1} * M_{projection}^{-1} * V_{clip}$$ (此处$M_{model} * V_{local}$ 才是世界坐标, V_{local} 只是局部坐标) 弄清楚坐标的转换关系之后通过调用 glut 库中的部分函数即可实现坐标系的转换。所用到的函数如下所示

函数名	void glReadPixels(GLint x, GLint y, GLsizei width, GLsizei height, GLenum format, GLenum type, GLvoid* data);
函数功能	从帧缓存里读取一个像素块
参数 x,y	指定从帧缓冲区读取的第一个像素的敞口坐标，此位置是矩形像素块的左下角
参数 width, height	指定像素矩形的尺寸
参数 format	指定像素的格式 GL_INDEX 单个颜色索引 GL_RGB 先是红色分量，再是绿色分量，然后是蓝色分量 GL_RED 单个红色分量 GL_GREEN 单个绿色分量 GL_BLUE 单个蓝色分量 GL_ALPHA 单个 Alpha 值 GL_LUMINANCE_ALPHA 先是亮度分量，然后是 Alpha 值 GL_STENCIL_INDEX 单个的模板索引 GL_DEPTH_COMPONENT 单个深度分量
参数 type	指定像素数据的数据类型
参数 data	返回的像素数据，是一个指针，指向存储图像数据的数组

函数名	glGetDoublev, glGetIntegerv
函数功能	获取投影矩阵和模型视图矩阵

函数名	gluUnproject()
函数功能	获取屏幕坐标对应的世界坐标
参数 winx, winy, winz	屏幕像素坐标，以左下角为起点
参数 modelMatrix	模型矩阵
参数 projMatrix	投影矩阵
参数 viewport	视口
参数 objx, objy, objz	返回值：世界坐标

函数实现:

```
1.  glm::vec3 getViewPos(int x, int y, GLfloat* pro, GLfloat* view)
2.  {
3.      //将 glm::mat4 类型转化为方法所需的数组类型
4.      GLfloat win_x, win_y, win_z;
```

```
    GLdouble object_x, object_y, object_z;
    int mouse_x = x;
    int mouse_y = y;
    win_x = (float)mouse_x;
    win_y = (float)viewPort[3] - (float)mouse_y ;
    //win_y = (float)viewPort[3] - (float)mouse_y;
    glReadBuffer(GL_BACK);
    glReadPixels(win_x, int(win_y), 1, 1, GL_DEPTH_COMPONENT, GL_FLOAT, &win_z);
    gluUnProject((GLdouble)win_x, (GLdouble)win_y, (GLdouble)win_z, modelView, projection, viewPort, &object_x, &object_y, &object_z);
    //使用 gluUnProject 方法将结果传入 object_x，object_y，object_z 中
    glm::vec3 p = glm::vec3(object_x, object_y, object_z);
    return p;
    //返回 vec3 类型
}
//转换鼠标位置到 3 维空间
```

3.3.9 其他

截屏并保存

截屏功能由 TakePicture 函数实现，按下 p 键可以截屏：

函数名	void TakePicture(void)
函数功能	截屏并保存，为了区分不同截图，将截图命名为现在时间。
实现思路	经查阅资料发现将截图保存为 bmp 格式文件较为方便，而简单起见，新建了一个空白的 bmp 文件，读取该文件的文件头作为保存 bmp 文件的文件头，然后使用 glReadPixels()函数读取(i, j)位置上的像素，然后写入即可。

关键代码如下所示：

```
//Allocate space for save file
PixelData = (GLubyte*)malloc(DataLength);
if (PixelData == 0)
    exit(0);
//read file head of dummy file
DummyFile = fopen("picture//test.bmp", "rb");
if (DummyFile == 0)
    exit(0);
//Get file name
char FileName[50] = "picture//CGProject_";
strcat(strcat(FileName, NowTime), ".bmp");
SaveFile = fopen(FileName, "wb"); //只写
if (SaveFile == 0)
    exit(0);

fread(BMP_Header, sizeof(BMP_Header), 1, DummyFile);
```

```
17. fwrite(BMP_Header, sizeof(BMP_Header), 1, SaveFile);
18. fseek(SaveFile, 0x0012, SEEK_SET);
19. i = wWidth;
20. j = wHeight;
21. fwrite(&i, sizeof(i), 1, SaveFile);
22. fwrite(&j, sizeof(j), 1, SaveFile);
23.
24. glPixelStorei(GL_UNPACK_ALIGNMENT, 4);
25. glReadPixels(0, 0, wWidth, wHeight,
26.     GL_BGR_EXT, GL_UNSIGNED_BYTE, PixelData);
27. // 写入像素数据
28. fseek(SaveFile, 0, SEEK_END);
29. fwrite(PixelData, DataLength, 1, SaveFile);
```

4 团队合作

4.1 团队分工

组员	负责部分	贡献占比
张立昱	● 机械臂建模、动画； ● 中期整合； ● 截屏-保存；	1/6
包德政	● 光照效果、物体材质、环境搭建； ● 合成碰撞检测； ● 最终流水线 demo 搭建；	1/6
徐梓凯	● 机械爪建模； ● 鼠标选取和鼠标放置； ● obj 文件导入导出	1/6
章沈柯	● 环境搭建、物体纹理； ● 漫游碰撞检测； ● nurbs 曲面； ● 最终流水线 demo 搭建	1/6
王方懿康	● 架构设计； ● 基类和通用接口设计； ● 物体变换效果； ● 保存与读取	1/6
钟沛沛	● 传送带和物体建模； ● 传送带运动； ● 最终流水线 demo 搭建	1/6

4.2 开发历程

时间	组员	完成任务
2021.12.02 - 2021.12.08	王方懿康	整体架构设计、基类设计、通用接口设计
	包德政	光照效果
	张立昱	完成机械臂的建模和贴图
	章沈柯	完成基本背景建模（包括地板、墙、门窗、书架、告示牌等），进行纹理贴图
	徐梓凯	机械爪建模
	钟沛沛	搭建传送带
2021.12.09 - 2021.12.18	王方懿康	完善机械臂基本类、传送带基本类、物体基本类
	包德政	完成聚光灯效果
	张立昱	将基本类整合到现有代码中
	章沈柯	完成门窗开关动画效果
	徐梓凯	完成机械爪抓取动画效果；修改物体漫游方式（修改为现在第一视角，类似 FPS 游戏的模式）
	钟沛沛	完成传送带建模并通过贴图实现动画效果；完成基本物体建模
2021.12.21 - 2021.12.30	王方懿康	完成类之间的绑定和数据（指传送带、机械臂和物体）的统一创建；
	包德政	完成光源变换（完成了光源的颜色、位置、开关的变化）
	张立昱	完成机械臂基本的动画效果，包括机械臂自身移动动画和对绑定物体的同步操作；完成对当前场景截图并且保存功能；协同搭建流水线
	章沈柯	探索碰撞检测实现方法，完成对视角不走出房间的碰撞检测；完成纹理变换功能，该功能体现在门口的告示牌上；完成 nurbs 曲面建模，体现在风扇上。
	徐梓凯	增加鼠标选取功能，添加了鼠标点击放置物体功能，放置为六棱柱；修改 shape 类，保证类下的坐标与世界坐标相同；探索将场景保存为 obj 文件并导出以及相对应的 obj 文件导入的功能
	钟沛沛	添加基本物体（添加了多面棱柱和多面棱台），整理了物体的 shape 类和传送带 conveyor 类，修改物体随传送带移动的逻辑，搭建基本的默认流水线
2022.01.17 - 2022.01.25	王方懿康	完成物体的变换效果（可以自由修改大小和颜色），保存与读取功能，存储介质设计
	包德政	协同完成搭建流水线 demo：完成物体合成碰撞检测，函数为 mooncakeCollision()
	张立昱	优化机械臂动画效果：增加一自由度，抓取更合理；增加对抓取位置物体的自动识别并绑定功能，实现自动抓取
	章沈柯	协同完成搭建流水线 demo：做好了饼皮、馅料、月饼、礼物盒皮和礼物盒；增加了 shape 中的 Type 属性与 RetType 函数
	徐梓凯	修正地板 y 轴坐标，修正屏幕坐标到世界坐标系转换函

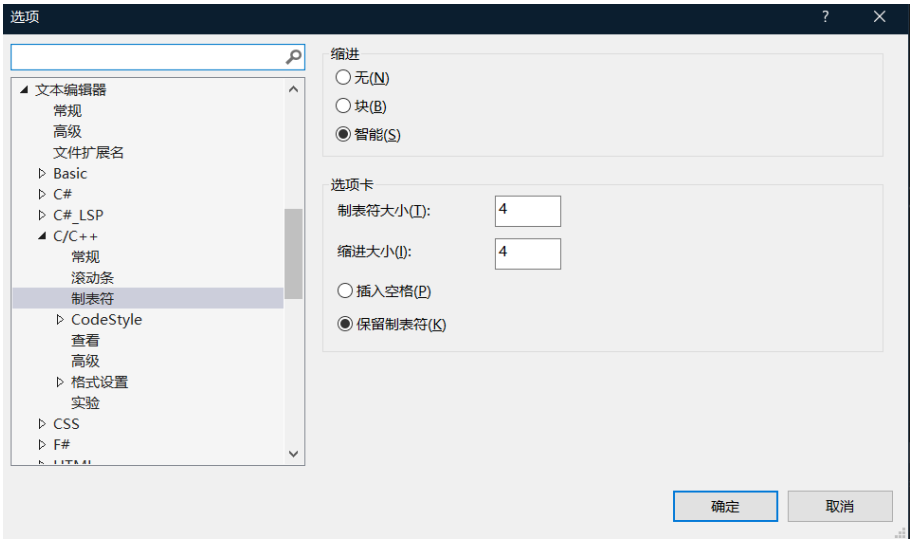
		数，增加鼠标选中的反馈效果
	钟沛沛	协同完成搭建流水线 demo：将各个元素整合，完成最终的流水线 demo
2022.01.26 - 2022.01.29	王方懿康	整理代码，完成报告
	包德政	
	张立昱	
	章沈柯	
	徐梓凯	
	钟沛沛	

4.3 编程规范

统一编码规范是团队协作中最基础的工作，是项目工程化的基石；保证代码风格的统一可以提高团队开发体验；配合编辑器自动格式化和代码检查器后还可以在开发过程中发现代码错误，极大的提升工作效率。

1. 编辑器通用配置

本次团队协作开发程序均使用 VS2019 进行开发，使用 VS2019 默认的文本编辑器编写代码，文本编辑器一些重要的配置如下图所示



编辑器采用默认的.editorconfig 文件

```
# Visual Studio 生成了具有 C++ 设置的 .editorconfig 文件。

[*.{c++,cc,cpp,cppm,cxx,h,h++,hh,hpp,hxx,inl,ipp,ixx,tlh,tli}]
vc_generate_documentation_comments = xml
```

采用 Visual Studio 风格的格式设置样式

所有源文件采用 UTF-8 格式进行编码

2. git 协作规范

本次团队协作通过 git 进行协作，工程中由于.vs 文件过大且并非为项目配置所必须的文件，需要通过配置忽略版本控制进行，具体方法为在项目根目录创建文件.gitignore，将 CGProject/.vs 通用规则写入其中

另外针对每个成员的使用情况，如果有其他目录或者文件需要忽略版本控制则会在相同文件下添加对应的条目。

团队成员在开发时会首先新建一个分支进行开发，在与其他团队成员沟通，进行冲突的解决后，会定期将稳定的工程同步到主分支上。

3. 编程注释规范

单行注释规范:

双斜线后，必须跟注释内容保留一个空格

可独占一行，前后不许有空行，缩进与下一行代码保一致

可位于一个代码行的末尾，注意这里的格式

样例如下

```
1.        // Good
2.  if (condition) {
3.     // if you made it here, then all security checks passed
4.     allowed();
5.  }
6.  var zhangsan = "zhangsan";   // 双斜线距离分号四个空格，双斜线后始终保留一个空格
```

多行注释格式

最少三行，格式如下

前边留空一行

```
1.  /*
2.   * 注释内容与星标前保留一个空格
3.   */
```

文件注释

```
1.        /*
2.   *@author: who are you
3.    *@date: when you write it
4.   *@description: the function of this file.
5.    */
```

函数头注释

函数头部应进行注释，一般情况下需要列出:

函数的目的/功能

输入参数

输出参数

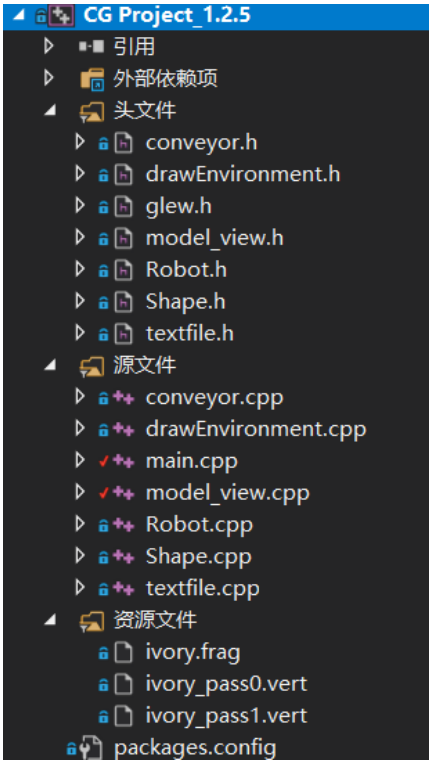
返回值

调用关系(函数、表)(非必须)

```
1.  /*
2.   * Function name：insert_hhistory
3.   * Description      : * Parameter      :
4.   * Return       :
5.   */
6.  int insert_hhistory(char* ipsql,risk host_level,int total,int t_id,char* t_uuid,char* ipaddr,long int end_time
    )
7.  {
8.     /*
9.      *   如果程序过于复杂，这里可以写明，具体算法和思路。
```

```
10.  */
11. }
```

4. VS 文件组织



头文件、源文件放到对应的筛选器下，其余的管线渲染文件、obj 文件放入资源文件筛选器下

4.4 合作总结

张立昱：

很荣幸作为大作业的组长，各位组员都很负责，在过程中提供了很多好的思路和想法，在讨论过程中很多次体会到了思想的碰撞，在大家的共同努力下，最终完成了大作业。作为一个外专业的学生，在这次作业中体会到了团队合作完成一个项目的感觉，感受到了最开始的系统设计和架构是很重要的，可以避免走很多弯路，过程中的交流和迭代对作品的最终质量也是很重要的。此外，在过程中对计算机图形学这门课程有了更加深刻的认识，包括对复杂物体的建模方法等；遇到了一些课程内没有学习过的知识，在大家的讨论和助教老师的帮助下得到了解决。另外，在组员的帮助下，我还学习了 github 的基本用法，对于项目后期的整合工作帮助很大。总结来说，这次大作业是一次愉快并且收获很大的体验。

徐梓凯：

这次计算机图形学大作业时间跨度较大，需要各个团队成员尽力协作，这次作业从总体设计思路开始一直到各自模块的完成以及最终各个模块的拼接，各个组员积极沟通，并且通过 github 进行协作，学习了多人协作中 github 许多的用法。

同时此次大作业中尝试运用课程上以及实验课上所学习的内容，加深了对理论课以及实验课教授的知识的理解，对可编程渲染管线的理解，以及对于图元操作的矩阵计算，三个

坐标系之间的转换等的理解, 结合自己生活中的体验, 对于一些 3D 建模、游戏等的认识提高了一个维度。

王方懿康：

我在这次的图形学大作业开发工作中，对 opengl 以及相关的图形学知识的理解愈加深刻，我的任务包含基类的设计与变换，涉及到坐标变化、旋转变化和尺寸变换，这是对图形学课堂知识的学习运用。同时，我也结合了其他课程的能力，运用软件工程与 OOP 相关概念模型来梳理整体软件架构，使得开发有条不紊。在总结中我想要特别感谢我的队友们，他们在开发中专业能力突出，沟通畅通，帮助我在 git 使用、程序设计等方面学会了很多。非常感谢张宏鑫教授和邹姗辰助教，他们在课程内容理解和环境配置等方面给予了我许多重要指导。

章沈柯：

这次有关智能工业的大作业对于我来说算是对于本学期图形学的一个系统梳理、应用和总结。如何将在课上学到的理论知识与实际编程相结合，从而达到更好的视觉效果，对于我来说是一个既富有趣味也充满挑战的课题。

本次我主要负责的内容时对于工厂环境和场景的搭建，其中也有表达相对复杂物体，例如门、窗等方面的需求，也尝试利用曲面进行建模。虽然在编程过程中遇到了一定困难，比如虽然大致上了解了 Nurbs 曲面的控制点概念，但如何在具体应用中调整控制点坐标，从而绘制出更加美观的物体依旧是一个难题。幸运的是，大体上问题都在查阅资料和咨询助教老师之后得到了解决，少部分解决繁琐或过分困难的，也找到了一些替代方案，在效果基本一致的前提下降低了难度。总体而言，在这次大程的编写中，如何提出需求，如何满足需求以及如何寻找替代方案达到需求这些方面，我都得到了锻炼。

不仅如此，由于本次是多人一组的大作业，与队友之间的磨合至关重要。无论是工程整体的架构，还是物块操控时的接口，都是需要事先对于对方的想法有一定了解，才能在编程中事半功倍。此外，多人代码的拼接是一项重要但困难的工作，在队友的帮助下，我学习了 github 的基本使用方式，这大大方便我们交接自己完成的代码，以及进行代码的拼接。

总而言之，在图形学这次大作业中，队友们都十分认真负责，而且给出的想法也颇有见地，在实现工程的过程中我不仅巩固了自己的理论知识，也收获了合作编程的宝贵经验，让我收获良多。

钟沛沛：

在这次图形学大作业中，我主要负责 shape 的实现、传送带构建与运动、搭建默认流水线等。我们将这学期学习到的图形学基本知识充分使用，并发挥创造力与想象力，完成了这一完整的作品。我体会到学以致用，用所学知识进行创造的快乐，并且加深了对课上所学知识的理解。我对团队合作也有了更深刻的体会，只有大家分工合作，才能得到这一工作量较大的作品。而在团队合作中，一些方法也必不可少，首先是代码框架的确定，可以使每个人的部分更容易整合；git 的使用也非常关键，可以进行版本控制，也便于合成每个人的工作；由于开发周期较长且每个人的部分不是完全独立的，每个人完成阶段性工作时写下完成的内容以及思路也非常重要，利于其他人了解代码，提高效率。最后，在合作的过程中，我的队友们能力都非常强，我们各取所长，解决了很多的问题，最终呈现出我们的大作业。

最后，感谢张老师的一学期的教导，以及助教姐姐在我编程有疑惑时的指点，希望以后能有机会接触更多图形学和建模的的知识。

包德政：

在图形学课程项目中，我主要负责了光照环境的设置、物体合成的碰撞检测以及协助大家梳理整合流水线 demo 的工作。在合作之中，很多次我都因为调试不出 bug 所在而感到有些灰心，尤其是一开始由于对光照理解不够透彻，导致对光照与纹理的结合没能正确处理。这也是我对 OpenGL 编程感触较深的一点，由于不能逐步调试，很多时候要依赖于显示效果与控制台输出才能启发自己发现 bug。

在编写过程中，除组内队友外，我也得到了包括邹珊辰助教学姐、专业朋友张沛全的帮助，感谢邹学姐提供的参考资料帮助我解决了光照与纹理结合的问题，感谢张沛全可以帮助我发现光照效果的 bug。

而在编写过程中，队友也给我留下了深刻的印象，梓凯扎实而优秀的编程能力，懿康有效而机智的解决方案，沈柯和沛沛的细致耐心，立昱的认真仔细，都给我带来了靠谱的感觉，在繁杂的编写过程中给我带来了信心。

总之，很开心能够最终完成“在 ZJU 学习 CG”这一人生小目标的打卡，也希望以后能够有机会接触到更加现代化的 CG 内容。感谢老师一学期的教导。

5 参考资料

1. OpenGL 的贴图模式 https://blog.csdn.net/u010309553/article/details/52886017
2. 屏幕坐标到世界坐标系变换[1] https://blog.csdn.net/u013253810/article/details/19906033
3. 屏幕坐标到世界坐标系变换[2] https://blog.csdn.net/u013378269/article/details/110108269?spm=1001.2101.3001.6650.1&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-1.pc_relevant_default&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-1.pc_relevant_default&utm_relevant_index=2