* F I N A L P R O J E C T -

**MINIMC**

BY刘秉贤 周益聪 王航宇 孔志林 王奕麟

# **CHAPTER 1 |** **项目介绍**

* 1. **项目简介**

MINIMC项目是由我们五位小组成员完成，以游戏《MineCraft》为原型设计和灵感来源、意在实现能够实现MineCraft游戏基本场景、功能和游戏逻辑的迷你版MC。

项目已经上传至<https://github.com/o-xygen/MiniMC>。

* 1. **小组成员及项目贡献**

|  |  |  |
| --- | --- | --- |
| 姓名 | 任务 | 占比 |
| 刘秉贤 | 分工、攥写工程文档并整合多个功能，完成MiniMC中的物理引擎，射线检测 | 33% |
| 周益聪 | 渲染相关，建立基础工程、完成场景方块渲染、优化至60帧渲染效果 | 27% |
| 孔志林 | 构建UI并合并，录像 | 19% |
| 王航宇 | 完成nurbs曲面建模模块，与刘秉贤一同尝试合并 | 14% |
| 王奕麟 | 整理文档、攥写报告 | 7% |
| 合计 |  | 100% |

* 1. **项目运行环境**

|  |  |
| --- | --- |
| **Programming Language** | C++ |
| **Project Framework** | OPENGL |
| **Operating System** | Windows 10 |
| **IDE** | Visual Studio 2019 |

# **CHAPTER 2 |** **游戏功能和逻辑设计**

* 1. **游戏功能设计**

**Physics**

1. 物件分刚体和非刚体，刚体包含墙、床、地、树、人、等等；非刚体包含一切可穿过物件，包括花、草、云、水等；

2. 没有碰撞反弹，只有逐帧碰撞检测；

3. 存在重力加速度；

4. 不存在摩擦力、空气阻力等概念，玩家移动便是直接移动；

5. 加入射线碰撞的检测，以获得某点打出的射线所触及的物件（用于检测玩家点击屏幕所操作的对象） 总之，可以将全部物件都处理成等大小的正方体。

**Rendering**

1. 暂时不制作天空盒，天空背景做蓝，根据实际测试的运行效率决定，使用贴图云朵或者方块堆叠云朵，可以不移动；

2. 物件基本都是简单的面片上贴图，如树、花、草、土等，建模以后编写导入功能来加载进场景；

3.渲染时，必须做到只渲染能看见的部分，包括可能看见的部分；此处需要一个模块裁定哪些大致需要渲染；

**Lighting**

因时间问题，不予完成光照与阴影等

**UI**

1. 始终显示在其他物件上方

2. 有暂停界面

**Others**

1. 截图功能；

2. 保存游戏数据、读取游戏数据。

* 1. **游戏逻辑设计**

**控制人物**

1. 移动：玩家可以通过位移按键控制人物移动（WASD键）；

2. 攻击或拆除：玩家按下鼠标左键控制人物攻击，屏幕中间的十字，既是视角中心，又是指向目标物体的“准星”；

3. 跳跃：玩家按下空格键可以从低处跳到高处平台上。

**动作**

1. 捡起物品：走进物品，在同一“格”中的物品会被立刻吸附到背包中，通过按键呼出UI面板展示；

2. 放置物品：玩家在装备可放置类物品时（土块，花，种子等），按下鼠标右键可以放置在准星处。

# **CHAPTER 3 |** **游戏实现**

**3.1 项目基本结构及其实现**

**场景绘制**

1. 提供API以在初始化时开始读入外存建模文件，并保存建模信息；

2. 提供API以对传入的三维数组绘制高效的地形地图。

**物理**

1. 负责存储所有要绘制物件的位置，是否为刚体

2. 存储玩家镜头角度信息

3. 检测鼠标点击时镜头对应的中心

4. 计算各自移动速度、起跳速度、掉落加速度，用于完成跳跃

5. 计算碰撞，分为与刚体碰撞、与非刚体碰撞

**游戏逻辑**

1. 存储逻辑判定有关的所有信息，与物理方面独立存储

2. WASD键控制人物移动，接管包括人物在内的所有移动信息，发送物件移动信息给物理模块

3. 提供API给物理；玩家按下鼠标左键控制人物对屏幕中间的十字对应物件做操作，由物理检测到鼠标点击后算出对应物件，调用逻辑模块对应API

4. 提供API给物理；物理在发生碰撞时回调逻辑，提供发生碰撞的两个物体的信息，用于拾取物体等事件

**UI**

可能要游戏逻辑部分专门与基础绘制通信，以完成UI绘制，可以搁置

* 1. **游戏实现进度**

**可初始化**

1. 创建工程，把依赖的glut glew丢进去；

2. 建立文档的文件夹。

**完成基础场景**

1. 用导入3dmesh的形式创建方块类，包括 泥土方块 带草皮的泥土方块 玻璃方块 大理石方块 箱子方块 沙土方块 混泥土方块 石砖方块 云朵方块

2. 用上述物件绘制一个简单场景，注意绘制顺序，选取哪些可以绘制、哪些不能绘制，将帧率控制在60以上，为后续模块留时间

3. 基础的移动能力，不考虑碰撞在内的物理特性、镜头方向改变

4. 不加入天空盒，背景留蓝

**完成较成熟的地图场景**

1. 存储地图 读取地图

2. 太阳贴图 平行光

3. 尝试NURBS曲?建模来绘制小物件，不行就算了

4. 导入花、树、小草、镐等物件

5. 与方块刚体物理碰撞 重力加速度 跳 行走 操作平滑舒适

6. 单一方向的射线碰撞，即镜头中心朝向的物件的计算

**完善细节**

1. 尝试天空盒 渐变的云朵以实现复杂材质效果图中云朵

2. 尝试泥土粒子特效 篝火粒子特效（好像不加分，可以不做

3. 游戏逻辑加入

4. UI加入

5. 光照阴影

# **CHAPTER 4 |** **算法代码实现**

**4.1 NURBS曲面**

我们为什么要使用NURBS曲面？

贝塞尔曲线的缺点是当我们增加很多控制点的时候，曲线变得不可控，其连续性会变差差。如果控制点很多（高阶曲线），当我们调整一个控制点的位置，对整个曲线的影响是很大的。要获得更高级的控制，可以使用GLU库提供的NURBS(非均匀有理B样条）。通过这些函数我们可以在求值器中调整控制点的影响力，在有大量控制点的情况下，依然可以产生平滑的曲线。

* **实现步骤**

以实现一个桌面的nurbs曲面为例：

1. 创建一个NURBS渲染对象 gluNewNurbsRenderer()

2. 调用相关的NURBS函数来修改曲线或曲面的外观 gluNurbsProperty

|  |
| --- |
| void \_nurbsInit() {  theNurb = gluNewNurbsRenderer();  gluNurbsProperty(theNurb, GLU\_SAMPLING\_TOLERANCE, 25.0);  gluNurbsProperty(theNurb, GLU\_DISPLAY\_MODE, GLU\_FILL);  } |

3. 定义表面，渲染NURBS gluBeginSurface gluNurbsSurface gluEndSurface

以汤勺勺柄上表面为例，定义表面

|  |
| --- |
| GLfloat handle\_spoon[4][4][3] = {  {  { -0.10, 1.5, 0}, { -0.05, 1.5, 0},  { 0.05, 1.5, 0}, { 0.10, 1.5, 0}  },  {  { -0.10, 0.85, 0.35}, { -0.05, 0.85, 0.50},  { 0.05, 0.85, 0.50}, {0.10, 0.85, 0.35}  },  {  { -0.25, -0.5, 0}, { -0.15, -0.5, 0},  { 0.15, -0.5, 0}, { 0.25, -0.5, 0}  },  {  { -0.15, -1.25, 0}, { -0.15, -1.5, 0},  {0.15, -1.5, 0}, {0.15, -1.25, 0}  }  }; |

绘制时传入上面的表面向量即可，下面是一个取一个16点的曲面并绘制的函数

|  |
| --- |
| void drawSurface(GLfloat surfaceVec[4][4][3]) {  GLfloat knots[8] = {0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0};  gluBeginSurface(theNurb);  gluNurbsSurface(theNurb, 8, knots, 8, knots, 4 \* 3, 3, &surfaceVec[0][0][0], 4, 4, GL\_MAP2\_VERTEX\_3);  gluEndSurface(theNurb);  } |

除此之外还有绘制4\*9\*3的向量，这是绘制由36个点定义的曲面时用到的表面绘制函数。

|  |
| --- |
| void drawSurface2(GLfloat surfaceVec[4][9][3], int x, int y, int z) {  int u, v;  GLint un = 30;  GLint vn = 30;  glMap2f(GL\_MAP2\_VERTEX\_3, 0, 1, y\*z, x, 0, 1, z, y, &surfaceVec[0][0][0]);  glEnable(GL\_MAP2\_VERTEX\_3);  glEnable(GL\_AUTO\_NORMAL);  glBegin(GL\_QUADS);  for (u = 0; u < un; u++) {  for (v = 0; v < vn; v++) {  glEvalCoord2f((GLfloat)v / vn, (GLfloat)(u+1) / un);  glEvalCoord2f((GLfloat)(v+1) / vn, (GLfloat)(u+1) / un);  glEvalCoord2f((GLfloat)(v+1) / vn, (GLfloat)u / un);  glEvalCoord2f((GLfloat)v / vn, (GLfloat)u / un);  }  }  glEnd();  } |

4. 销毁NURBS渲染对象 gluDeleteNurbsRenderer();