Project report for the CG 100433 course

Project Title

Crazy Fruit（限时闯关游戏）

Team member

1752657 李卓凡、1752592 方思宇、1850182 魏然、1852921 户英豪、1854167 徐思琪、1851801 刘子云

Abstract

本项目实现了一个卡通风格的限时闯关游戏，灵感来源于糖豆人游戏中的一个关卡。首先我们对场景、角色和 UI 进行了设计，充分利用了三维建模的技能和网上的资源，打造出了丰富多彩的视觉效果。之后使用 bullet 物理引擎，实现了人物和场景的碰撞，并模拟出了障碍物漂浮在水中的动力学效果。为了让人物的动作更加生动，我们制作了骨骼动画并成功读取和渲染。接着加入了水面效果的实现，让水面呈现出流动的效果，并带有透明和反光，显得更加真实。最后添加了 UI，对游戏中的各种细节加以优化。总体而言本项目综合性较强，涵盖了场景渲染、骨骼动画、物理引擎和水面模拟等多种技术。

Motivation

本项目的灵感来源于糖豆人游戏中的一个关卡，为了降低难度，对原关卡进行了适当的简化，将人物和障碍物的直接碰撞简化为了载具和障碍物的碰撞，从而避免了复杂的骨骼动画转换。简化设计后，本小组希望能够实现一个完整度高的、既有技术含量又有艺术性的游戏。

The Goal of the project

1. 实现游戏场景中的物理效果
2. 实现骨骼动画的渲染
3. 实现水面的模拟
4. 实现场景、人物和障碍物的建模
5. 实现场景的渲染，包括天空盒、水面、UI 等特效
6. 实现完整的游戏流程

The Scope of the project

1. 骨骼动画和物理引擎不会产生交互，仅是根据物理引擎计算出的结果变换位置。
2. 物体和水面不会产生拖尾、波浪的交互效果，水体的模拟是独立的。

Involved CG techniques

1. 骨骼动画
   制作、读取、渲染
2. 物理引擎
   漂浮物体的动力学模拟
3. 水面模拟
   波浪、颜色

Project contents

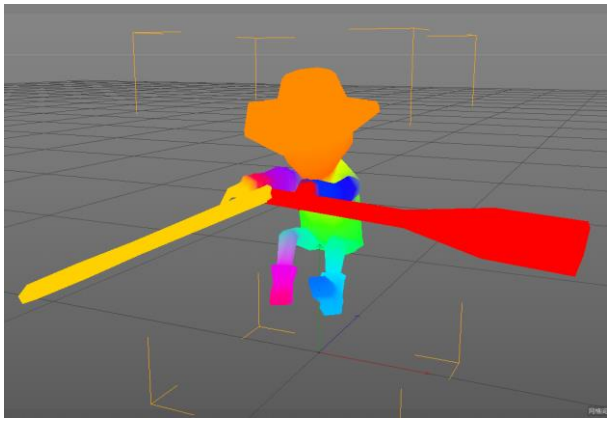
一个限时的闯关游戏，将实现以下功能
1. 游戏状态转换和胜负判定
2. 玩家角色的控制和角色的骨骼动画
3. 玩家与障碍物的交互以及障碍物之间的碰撞
4. 场景、天空盒以及水面的渲染

Implementation

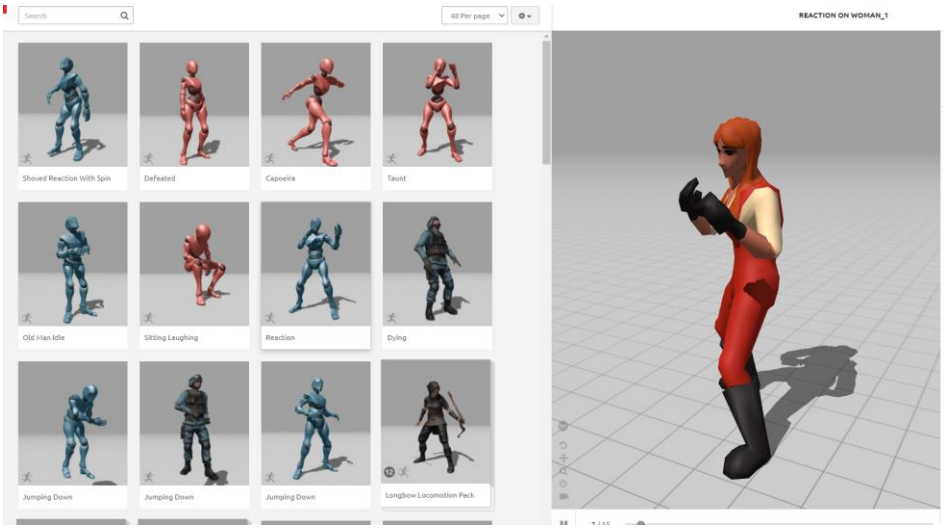
一、骨骼动画

1. 制作

游戏中包含四个骨骼动画——划船、鼓掌、欢呼和捂脸。其中由于划船的动作找不到现成的资源，故使用 Cinema4D 软件进行制作。



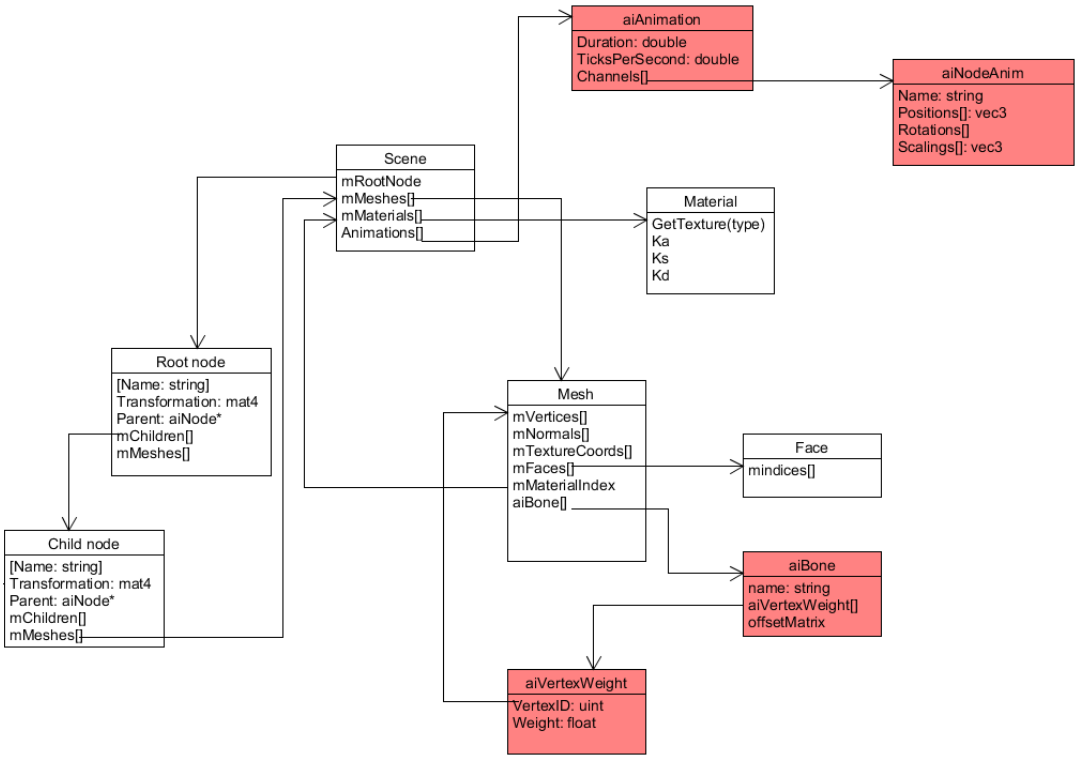
另外三个骨骼动画利用了网站 www.mixamo.com 上的资源。



2. 读取

使用的第三方库：assimp。

数据结构如下，其中以 Scene::mRootNode 为根节点的树存储了一颗骨骼树。骨骼与顶点的关系存储在了 Mesh 类的 aiBone 成员中，包含了骨骼对其影响的顶点的权重信息。骨骼动画的变换数据存放在了 aiAnimation 类中，为每个骨骼记录了其关键帧的相对变换矩阵。



对以上数据结构进行处理，这里设定一个顶点最多受到四个骨骼的影响，因此将该四个骨骼的 ID 以及权重存储在顶点类中，作为顶点属性，便于顶点着色器的编写。

3. 渲染

对骨骼动画类输入一个时间参数，递归地计算每个骨骼在世界坐标系下的变换矩阵。由于动画序列中只保存了关键帧的信息，因此需要进行线性插值。得到的变换矩阵还需要乘上父节点的变换矩阵，才能得到世界坐标系下的变换矩阵。这部分代码如下：

```
//scaling
aiVector3D scaling_vector = calcInterpolatedScaling(p_animation_time, node_anim);
aiMatrix4x4 scaling_matr;
aiMatrix4x4::Scaling(scaling_vector, scaling_matr);
//rotation
aiQuaternion rotate_quat = calcInterpolatedRotation(p_animation_time, node_anim);
aiMatrix4x4 rotate_matr = aiMatrix4x4(rotate_quat.GetMatrix());
//translation
aiVector3D translate_vector = calcInterpolatedPosition(p_animation_time,
node_anim);
aiMatrix4x4 translate_matr;
aiMatrix4x4::Translation(translate_vector, translate_matr);
node_transform = translate_matr * rotate_matr * scaling_matr;
aiMatrix4x4 global_transform = parent_transform * node_transform;
```

着色器部分，所有骨骼的变换矩阵通过 uniform 传递给顶点着色器，记为 bones[]。顶点着色器中可以获取该顶点受到哪些骨骼的影响以及其权重。这里设定一个顶点最多受到四个骨骼的影响。将该顶点对应的四个骨骼变换矩阵加权相加，并乘以顶点的原有位置，就可以得到顶点受到骨骼影响后的位置。这部分代码如下:

```
mat4 bone_transform = bones[bone_ids[0]] * weights[0];
bone_transform += bones[bone_ids[1]] * weights[1];
bone_transform += bones[bone_ids[2]] * weights[2];
bone_transform += bones[bone_ids[3]] * weights[3];
vec4 boned_position = bone_transform * vec4(aPos, 1.0);
```

二、物理引擎

我们使用的是 bullet 库作为物理引擎。我们完成了物理世界的创建、3D 模型的导入、静态动态碰撞体的创建、水的浮力、推力以及凝滞阻力的模拟、船体的控制以及船体被撞歪后的矫正等工作。

1. 模型导入

在 3D 模型的导入时，我们读取和处理的是通用的 obj 文件，关于 obj 文件的格式就不再赘述，我们将读出的数据转化为对应的 mesh 结构的数据。

物理世界的创建通过一些较固定的流程来完成创建，在物理世界创建时，可以进行重力的相关设置。

2. 静态物体创建

静态物体的创建我们采用的是静态三角片面模型 btBvhTriangleMeshShape，创建通过一个相关的类：btTriangleMesh 来辅助创建，btTriangleMesh 通过类内函数 addTriangle 传入三角形的三个顶点来完成三角片面的添加，完成所有片面的添加后，通过该类来完成静态三角片面模型的创建，同时需要设置质量为 0，还有其他的一些相关的参数设定，包括物体的位置、旋转、滑动摩擦力、滚动摩擦力、弹性系数、初始冲量等信息的设定，创建完成后将物体加入到世界中。

```
    btTriangleMesh* triangle_mesh = new btTriangleMesh();
```

```
for (int i = 0; i < obj_info.objPoints.size(); i += 3) {
    btVector3 vertex_1 = obj_info.objPoints[i];
    btVector3 vertex_2 = obj_info.objPoints[i + 1];
    btVector3 vertex_3 = obj_info.objPoints[i + 2];
    triangle_mesh->addTriangle(vertex_1, vertex_2, vertex_3);
}
btBvhTriangleMeshShape* triangle_mesh_shape = new btBvhTriangleMeshShape(triangle_mesh, true);
btBvhTriangleMeshShape* collisionShape = new btBvhTriangleMeshShape(*triangle_mesh_shape);
```

3. 动态物体创建

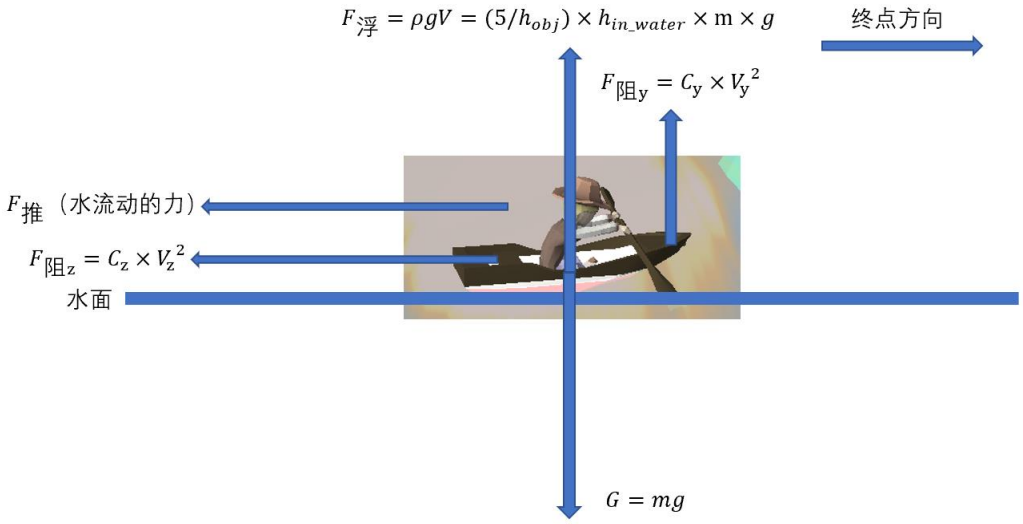
动态物体的创建，我们采用的是凸体模型的类：btConvexHullShape，来构建一个凸体。构建凸体的过程相对来讲较为简单，只需要把相关的顶点传入即可，通过类内函数 addPoint 来添加顶点，也可以在对顶点进行优化之后再进行相关的创建。碰撞体创建完成后，对其相关信息进行设定，质量需要为正数，还有其他的相关参数的设定，然后将动态物体加入到世界中。

```
btConvexHullShape* unoptimized_hull = new btConvexHullShape();
for (int i = 0; i < obj_info.objPoints.size(); ++i)
    unoptimized_hull->addPoint(obj_info.objPoints[i]);

btShapeHull* hull_optimizer = new btShapeHull(unoptimized_hull);
btScalar margin(unoptimized_hull->getMargin());
hull_optimizer->buildHull(margin);
btConvexHullShape* hull = new btConvexHullShape(
    (btScalar*)hull_optimizer->getVertexPointer(),
    hull_optimizer->numVertices());
btCollisionShape* collisionShape = new btConvexHullShape(*hull);
```

4. 水的物理模型

在游戏中，游戏人物是在水面上进行移动的，因此需要在物理世界中构建水的物理模型。要构建水的物理模型，就是要设定水给物体的力。当物体与水接触之后，受力情况如下图所示：



具体介绍如下：

1) 重力

根据上述设定的物理世界的全局的重力系数 g，物体会受到朝向负 y 轴的重力

$$G = mg$$

2) 浮力

水会给在水中的物理一个浮力，浮力的方向朝向正 y 轴。浮力的计算公式为

$$F_{浮} = \rho g V$$

但是计算物体的体积 V 是比较困难的事情，于是我们将整个式子转化为了，当物体的高度的五分之一没入水中之后，此时物体所受到的浮力就等于重力。即当物体高度的五分之一进入水中之后就受力平衡。

因此浮力的计算式子改为：

$$F_{浮} = \rho g V = (5/h_{obj}) \times h_{in\_water} \times m \times g$$

其中h_{obj}为物体的高度，h_{in_water}为物体进入水中的高度。h_{obj}由物体各顶点的 y 轴最大值减去 y 轴最小值得到；h_{in_water}由物体中心点位置的 y 轴大小与水面的 y 轴大小之差得到。

3) 水的凝滞阻力

当物体在水中与水有相对速度时，物体会受到与速度方向相反的来自水的凝滞阻力。该凝滞阻力与物体的速度的平方成正比，因此计算的公式为：

$$F_{凝} = \begin{pmatrix} F_x \\ F_y \\ F_z \end{pmatrix} = -(c_x, c_y, c_z) \begin{pmatrix} {v_x}^2 \\ {v_y}^2 \\ {v_z}^2 \end{pmatrix}$$

其中c_x, c_y, c_z为在各个方向上的凝滞阻力系数，v_x, v_y, v_z为物体在各个方向上的速度。

凝滞阻力的实现，可以让物体在水中有速度时，会因为该阻力而被迅速减速，而不至于一直在水中左右漂动以及上下浮动。

4) 水的推力

游戏中对于水的设定是水在一直流动，因此水流会不停的给物体推力，直至物体的速度与水流速度一致，即物体与水流不存在相对速度为止。这个力为一个定值的力，不会随着物体与水的情况的变化而发送变化，因此只要当物体的速度小于水流速度或物体速度与水流速度相反时，则物体就会受到该推力。

上述四个力通过计算之后，全部施加到每个在水面的物体上，通过 bullet 库提供的函数，如下所示：

```
body->applyCentralForce(buoyancy_force + water_drag_force + water_push_force);
```

5. 人物移动的控制

人物的移动同样通过力与速度来实现。当我们按下上左右箭头键的时候，则会对于人物的移动进行控制。人物的前后移动通过给人物一个向前或向后的推力来实现，该力为定值。而人物的左右转动，通过给人物一个向左或向右的旋转速度来实现。由于是给速度，会由于惯性继续转动，这样不好控制，因此我们在每一帧计算之后都将物体的旋转速度置为 0。这样我们每次控制物体的旋转就只会让该速度作用于一帧，起到了模拟旋转的力的效果。

6. 船体旋转角的矫正

关于船体被撞歪之后的矫正问题，我们对船体的 roll 角、yaw 角设定了阈值，当船体与

其他物体发生碰撞导致船体的 roll 和 yaw 超出了所设定的阈值，我们就会对其进行矫正，采取的方式是改变船体在 roll 和 yaw 上的角速度，使船体以一定的、递减的角速度回正，这样就可以完成关于船体撞歪后的矫正工作。

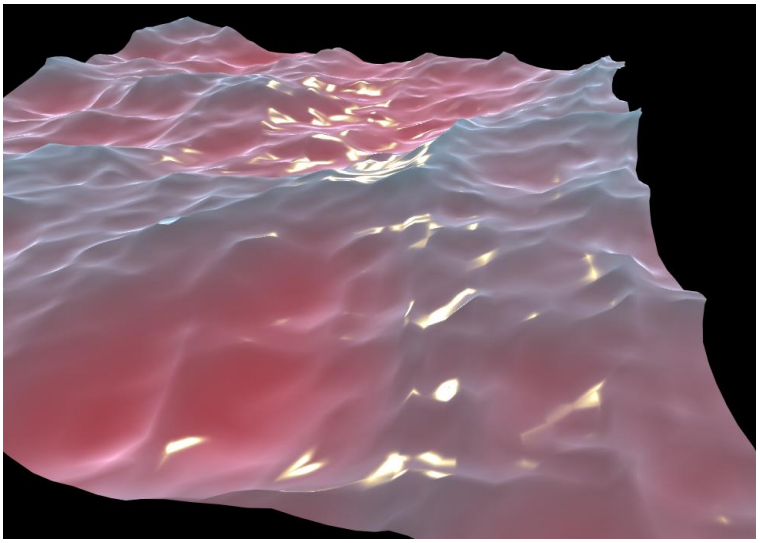
三、水面模拟

1. 水面着色器

```
vec3 shallowColor =  vec3(86, 170, 252) / 255.0;
vec3 deepColor =  vec3(74, 152, 252) / 255.0;

float relativeHeight; // from 0 to 1
relativeHeight = (FragPos.y - heightMin) / (heightMax - heightMin);
vec3 heightColor = relativeHeight * shallowColor + (1 - relativeHeight) * deepColor;
vec3 combinedColor = ambient + diffuse + heightColor + reflectColor;
color = vec4(combinedColor, 0.7f);
```

这里重点在不同高度的波的颜色不同，从波的最高处到最低处呈现蓝→红的变化



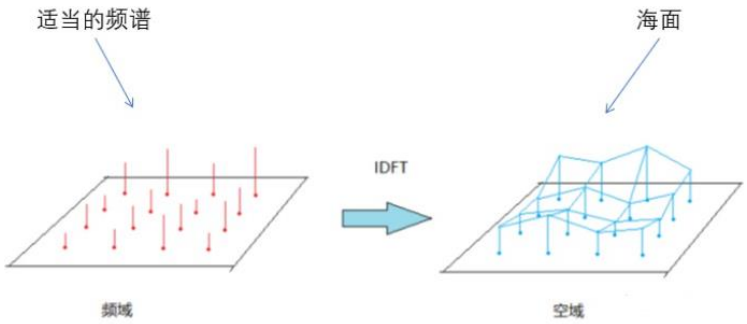
2. 水面波浪效果[5]

论文中提出了两种创建海洋表面的实际方法。一种是基于水结构和运动的简单模型，另一种是根据实验证据将大量振幅相互关联的正弦波相加。第二种方法利用快速傅里叶变换（fft）技术进行求和，我们这次项目中也采用了这种方法。

具体的实现如下:

如果将开放海域的波浪高度看作定义在 XZ 平面上的空域信号 $h(x,z)$，根据经验，这个信号天然地就很接近大量正弦信号的叠加。如果我们知道了其频谱 $\tilde{h}(\omega_x,\omega_z)$，则使用傅里叶逆变换，就可以求出$h(x,z)$，即得到海面的高度场。

由于计算机不能处理连续或无限的事物，所以海面模拟里用的傅里叶逆变换是离散傅里叶逆变换（IDFT）。



海面高度由下面 IDFT 给出：

$$h(\vec{x},t)=\sum_{\vec{k}}\tilde{h}(\vec{k},t)e^{i\vec{k}\cdot\vec{x}}$$

其中：

$\vec{x}=(x,z)$ 为空域坐标，$\vec{k}=(k_x,k_z)$ 为频域坐标，k_x,k_z 均为频率，相当于前文中的 ω_x,ω_z

$\tilde{h}(\vec{k},t)$ 是频谱，较前文多了个参数 t，表示此频谱会随时间变化，相应地高度函数就也变成随时间变化的 $h(\vec{x},t)$ 了。

频谱 $\tilde{h}(\vec{k},t)$ 的计算源于海洋统计学，通常采用的是菲利普频谱，具体公式如下：

$$\tilde{h}(\vec{k},t)=\tilde{h}_0(\vec{k})e^{i\omega(k)t}+\tilde{h}_0^*(-\vec{k})e^{-i\omega(k)t}$$

其中，$\tilde{h}_0^*$ 表示 $\tilde{h}_0$ 的共轭复数，k 表示 $\vec{k}$ 的模。

$\omega(k)=\sqrt{gk}$，g 为重力加速度

代码实现如下：

```
inline float Wave::func_omega(float k) const
{
    return sqrt(g*k);
}
```

$$\tilde{h}_0(\vec{k})=\frac{1}{\sqrt{2}}(\xi_r+i\xi_i)\sqrt{P_h(\vec{h})}$$

其中 ξ_r 和 ξ_i 是相互独立的随机数，均服从均值为 0，标准差为 1 的正态分布。

代码中的实现如下：

```
inline complex<float> Wave::func_h_twiddle_0(vec2 vec_k)
{
    float xi_r = normal_dist(generator);
    float xi_i = normal_dist(generator);
    return sqrt(0.5f) * complex<float>(xi_r, xi_i) * sqrt(func_P_h(vec_k));
}
```

完成以上内联函数的构造后，对频谱 $\tilde{h}(\vec{k},t)$ 的计算如下：

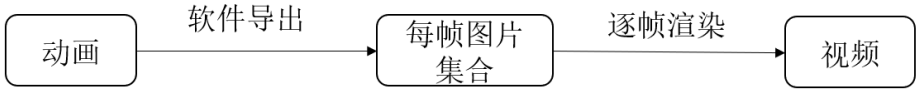
```
inline complex<float> Wave::func_h_twiddle(int kn, int km, float t) const
{
    int index = km * N + kn;
    float k = length(K_VEC(kn, km));
    complex<float> term1 = value_h_twiddle_0[index] * exp(complex<float>(0.0f,
func_omega(k)*t));
    complex<float> term2 = value_h_twiddle_0_conj[index] * exp(complex<float>(0.0f, -
func_omega(k)*t));
    return term1 + term2;
}
```

四、UI 界面渲染

1. 视频图片渲染

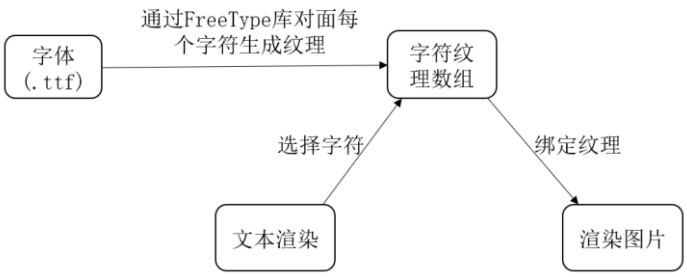
对于图像的渲染，我们使用了 stb_image.h 图像文件加载库。这个库的作用是可以将输入的图片以指定的方式（如 RGA、RGBA）读取后输出为纹理数据。根据生成的纹理数据我们就可以生成 OpenGL 纹理，再绑定在矩形图像上就能实现图片渲染了。为了使图片静态显示在窗口中，我们可以使用和窗口大小一致的投影矩阵，这样就能固定图片相对窗口的位置了。

在图片渲染的基础上，只要把视频按一定帧率导出为图片，再在每次渲染时按帧率依次渲染图象就能实现视频的渲染了。

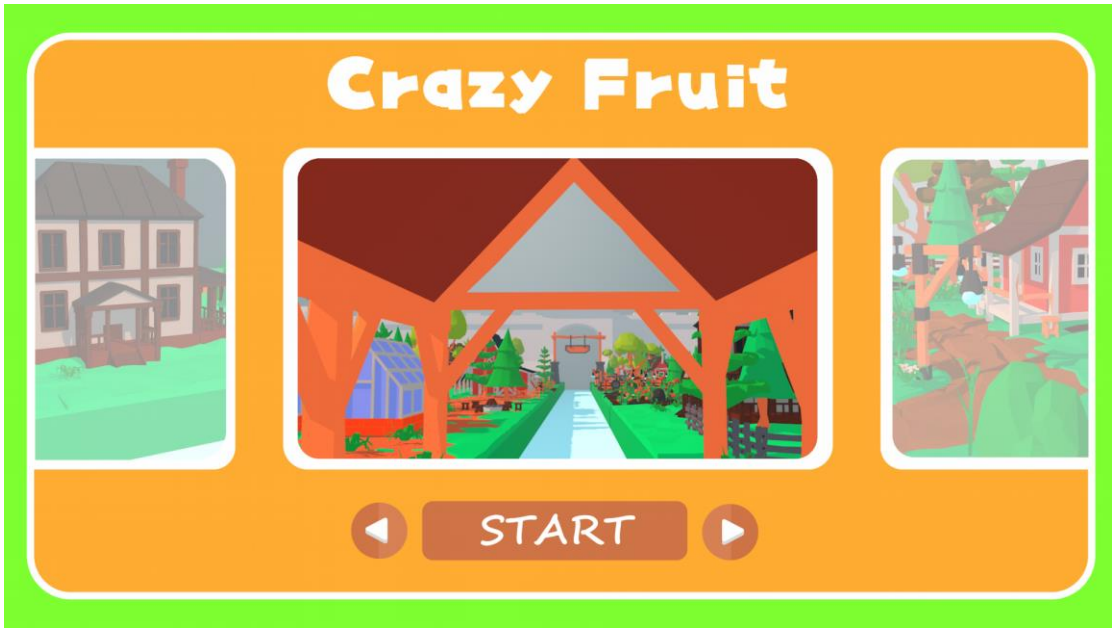


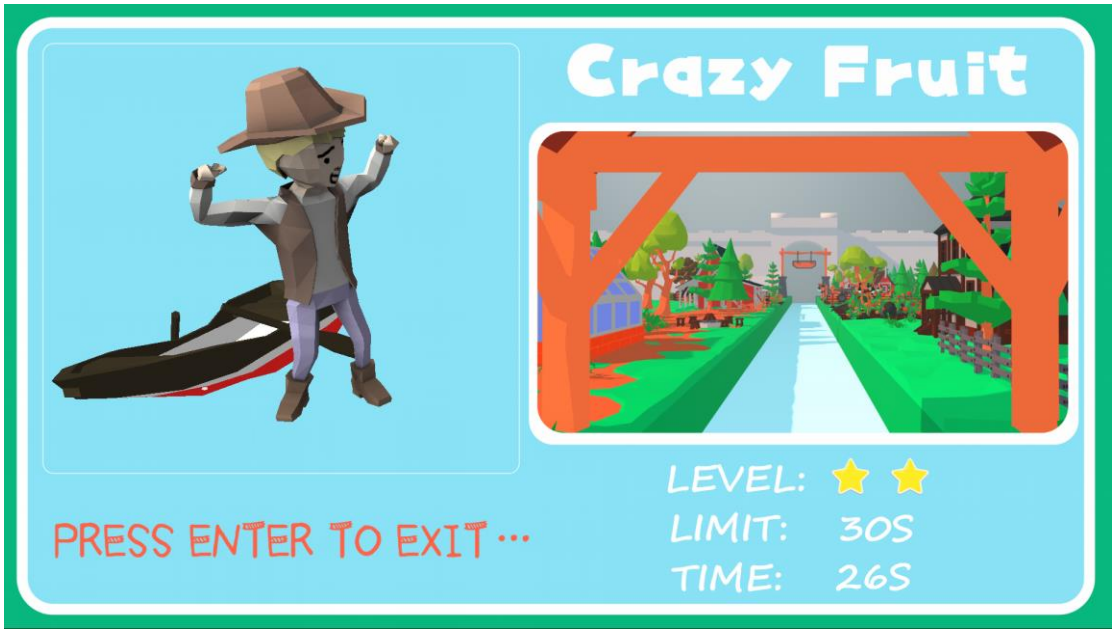
2. 文本渲染

对于文本的渲染，我们使用了 FreeType 库。FreeType 是一个能够用于加载字体并能将他们渲染到位图的开发库。当我们需要一种字体是，我们通过 FreeType 库读取.ttf 字体文件，并对前 128 个 Ascii 字符进行纹理生成，将他们保存在自定义数据类型中，在进行文字渲染既可以调用这些纹理，再按照类似图片渲染的方式进行显示了。



Results





Roles in group

魏然、户英豪（贡献比各 18%）：物理引擎的整合、各种动力学效果的仿真
李卓凡（贡献比 18%）：场景的设计、骨骼动画的设计和渲染、游戏主流程的实现
方思宇（贡献比 16%）：UI 界面的设计和渲染、障碍物的建模
徐思琪、刘子云（贡献比各 15%）：水面的模拟和渲染

References

[1] bullet 物理引擎与 OpenGL 结合 导入 3D 模型进行碰撞检测 以及画三角网格的坑 https://www.cnblogs.com/DOMLX/p/11681069.html
[2] https://github.com/bulletphysics/bullet3
[3] https://blog.csdn.net/pizi0475/article/details/9955611?utm_source=blogxgwz
[4] https://gamedev.stackexchange.com/questions/99045/how-to-load-a-level-for-use-with-the-bullet-physics-library
[5] Tessendorf J. Simulating ocean water[J]. Simulating nature: realistic and interactive techniques. SIGGRAPH, 2001, 1(2): 5.