动态三视图的实现与 VR 演示

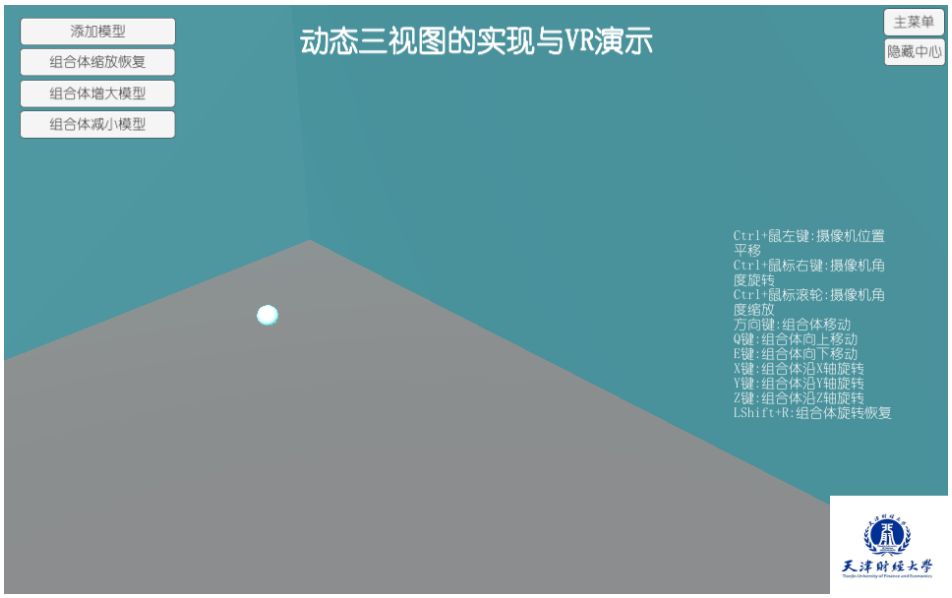
----实验步骤

（一） 动态三视图观察实验步骤

（请下载 3D.zip 解压缩后，使用 Edge 或 Firefox 浏览器双击 index.html 打开）

（1）观察现有模型

1.进入动态三视图的实现与 VR 演示网页并点击"View"按钮。



2. 按住 LCtrl+鼠标右键,旋转视角。
3. 按住 LCtrl+鼠标左键拖动屏幕。
4. 按住 LCtrl+鼠标滚轮缩放视角。
5. 经过 234 三步将屏幕中心对准白色球体(对象中心)并显示合适大小。
6. 点击添加模型按钮,生成一个模型，按住左 shift 键，滚动鼠标滚轮，选择正方体模型。
7. 按下键盘上的方向键模型移动至合适位置。
8. 按下键盘上的 V 键，将模型放大至最大倍数。
9. 按下键盘上的 X 键，将模型沿自身 X 轴旋转任意角度。
10. 按下键盘上的 Y 键，将模型沿自身 Y 轴旋转任意角度。
11. 按下键盘上的 z 键，将模型沿自身 z 轴旋转任意角度。
12. 按下 Shift+R 键恢复模型初始状态。
13. 重复 7。
14. 按下键盘上的 B 键，将模型减小至最小倍数。
15. 重复 8-10。
16. 按下鼠标左键将模型放置成功。
17. 按下键盘上的 X 键，将模型沿白球中心点 X 轴旋转任意角度。
18. 按下键盘上的 Y 键，将模型沿白球中心点 Y 轴旋转任意角度。
19. 按下键盘上的 z 键，将模型沿白球中心点 z 轴旋转任意角度。
20. 观察放置的模型以及其三视图。
21. 重复以上步骤，再添加一个模型组成组合体。
22. 观察组合体。

（2）三棱锥模型绘制实验

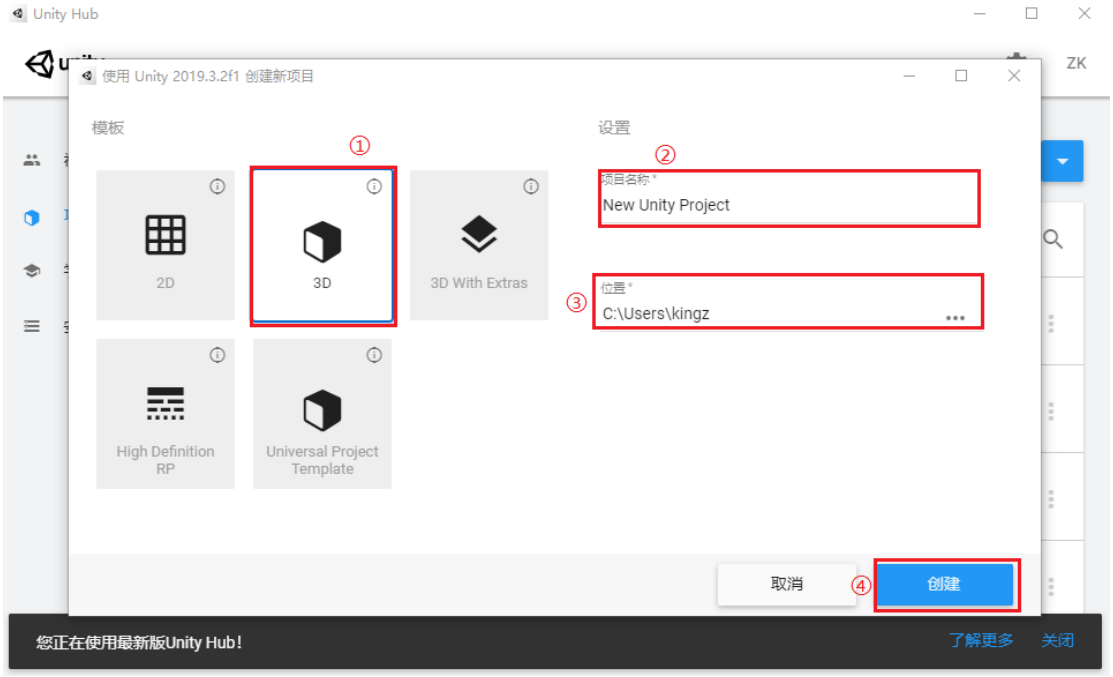
1. 进入动态三视图的实现与 VR 演示网页并点击"Create"按钮。

2. 输入顶点个数（4）
3. 输入三角形索引组数个数（8）
4. 依次输入以下坐标（-2，-1，-2）（2，-1，-2）（0，-1，2）（0，2，0）
5. 依次输入以下索引（0，1，2）（0，2，1）（0，1，3）（0，3，1）
（1，2，3）（1，3，2）（2，0，3）（2，3，0）
6. 重复实验（一）6-11、16 将刚才绘制的三棱锥放置成功
7，重复实验（一）17-20。
7. 自己写出超过 10 个顶点的复杂模型，并以三个顶点为一组写出所有的三角形索引
8. 点击”添加 Mesh“按钮，将 7 中的点和索引按照 2-5 的步骤方法将新 Mesh 创建成功
9. 重复实验（一）21-22
(3) 自定义实现模型绘制
仿照（2）自定义顶点与索引实现正方体的绘制。

（二） 虚拟仿真实验本地实验步骤

（请下载 Assets.zip，并安装 Unity3D 软件）

1：打开 Unity 软件，新建一个 3D 项目，命名为 VR_Mesh_Test，然后选择项目的路径，完成后点击创建，等待创建成功。



2：进入 Unity 开发界面，熟悉其各个面板的功能与位置。
①Scene：场景视图是一个可交互的沙盘。可以使用它来选择并在场景中定位所有的游戏对象。
②Game：游戏视图是显示游戏最终运行效果的，通过单击工具栏中的播放按

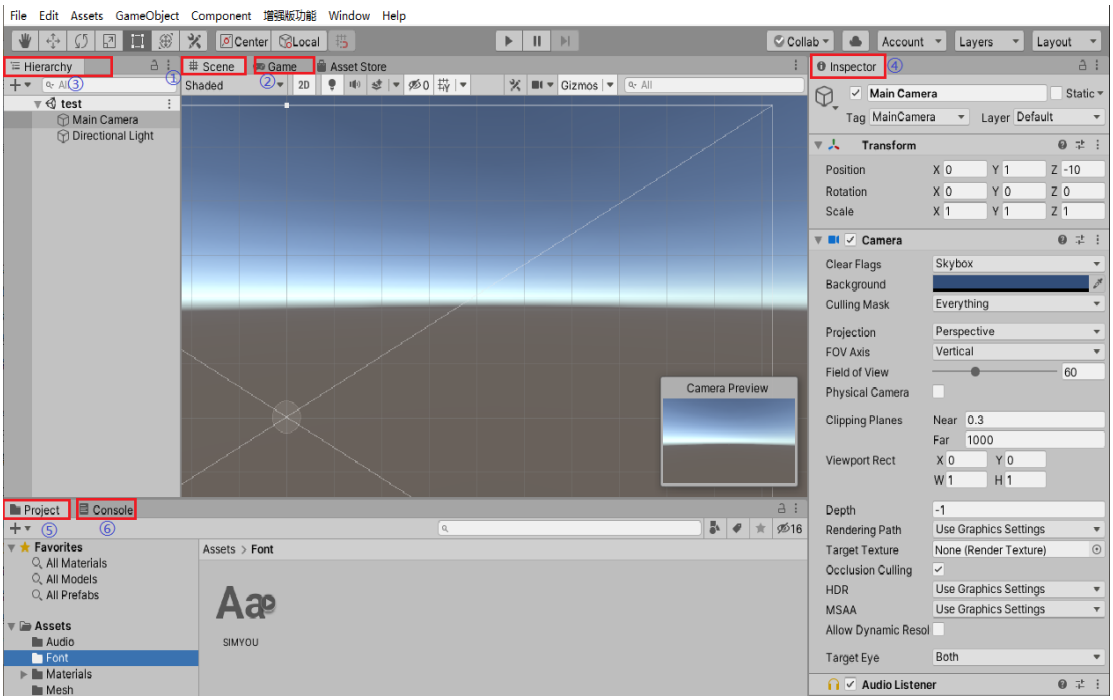
钮即可在游戏视图中进行游戏的实时预览以进行游戏开发和调试。

③Hierarchy：“层次”窗口按层次顺序列出我们打开的场景中的所有游戏对象，在这个窗口中可以创建一些简单的游戏对象、摄像机、文本显示等素材。

④Inspector：检视面板显示当前选中物体的基本信息，也显示它所包含的组件和组件的属性。

⑤Project：包含了整个项目所有的资源和管理（包括项目用到的插件，脚本，材质，模型等）。

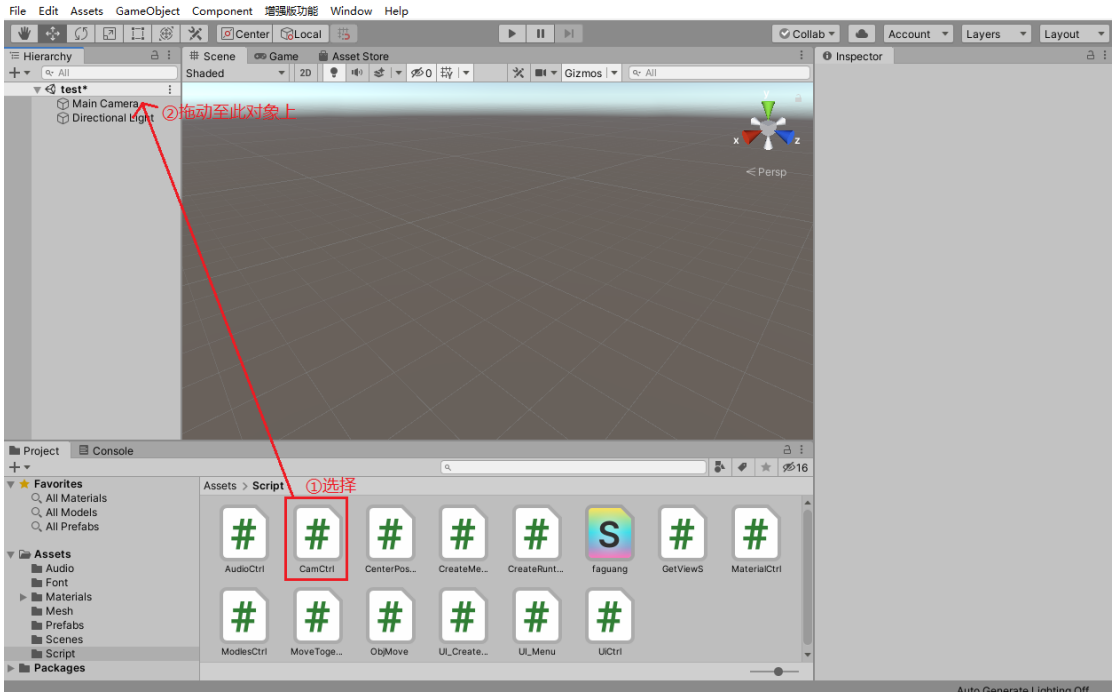
⑥Console：控制台主要是用来输出日志进行调试，可自定义输出信息。



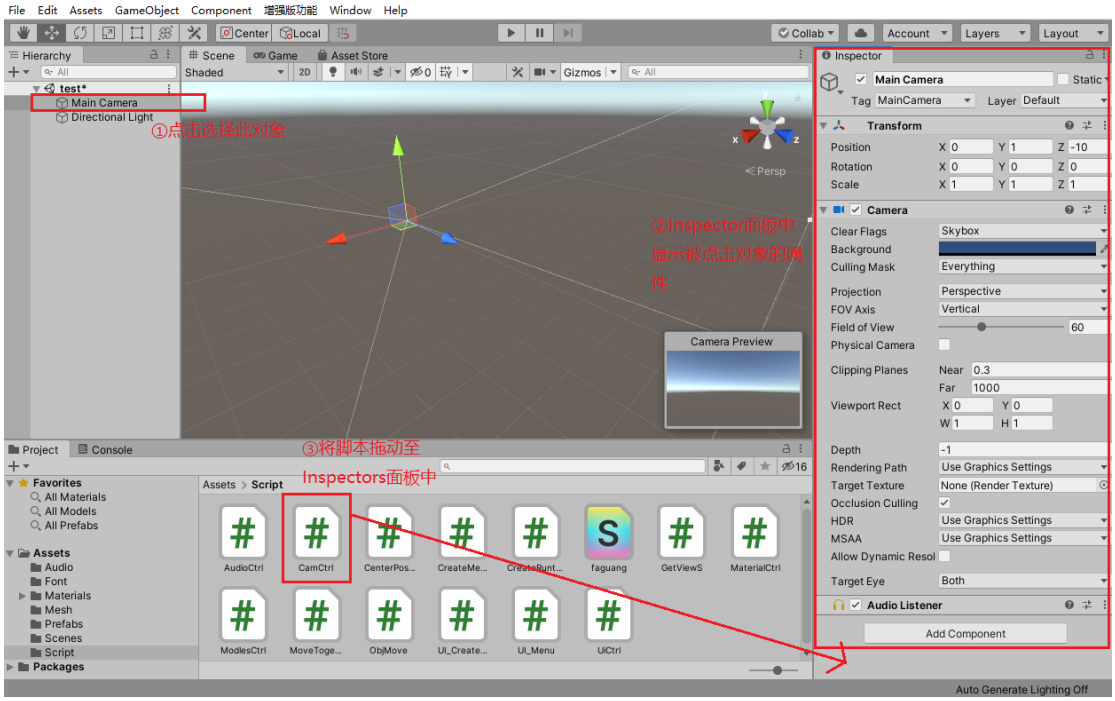
3. 打开 XX 路径下的压缩包，解压缩至刚才新建项目路径的 Assets 目录下，并刷新 Unity。

名称	修改日期	类型	大小
Audio	2020/3/15 23:18	文件夹	
Font	2020/3/14 22:32	文件夹	
Materials	2020/3/16 19:47	文件夹	
Mesh	2020/3/7 14:32	文件夹	
Prefabs	2020/3/15 13:04	文件夹	
Scenes	2020/3/18 17:57	文件夹	
Script	2020/3/17 14:03	文件夹	
Audio.meta	2020/3/5 12:16	META 文件	1 KB
Font.meta	2020/3/14 22:29	META 文件	1 KB
Materials.meta	2020/3/5 12:15	META 文件	1 KB
Mesh.meta	2020/3/7 14:32	META 文件	1 KB
Prefabs.meta	2020/3/11 13:25	META 文件	1 KB
Scenes.meta	2020/3/4 20:01	META 文件	1 KB
Script.meta	2020/3/5 12:06	META 文件	1 KB

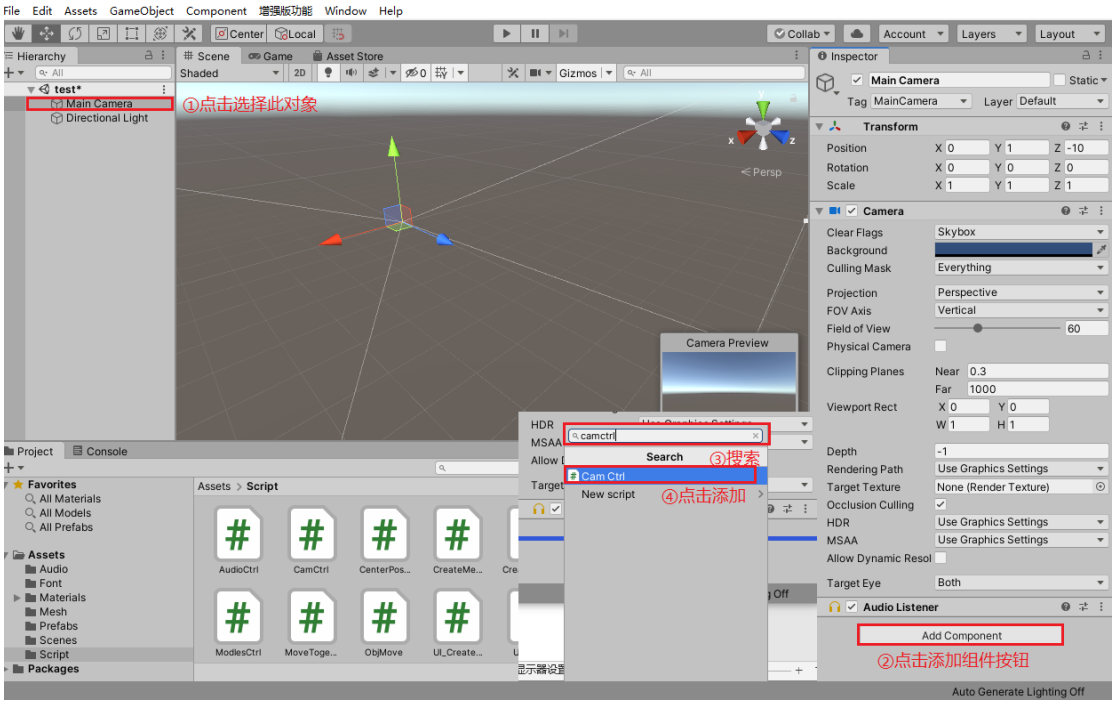
4. 为摄像机添加移动脚本。方法①：打开 Project 窗口下的 Script 文件夹，找到 CamCtrl 脚本，将脚本拖动至 Hierarchy 下的 Main Camera 对象上。



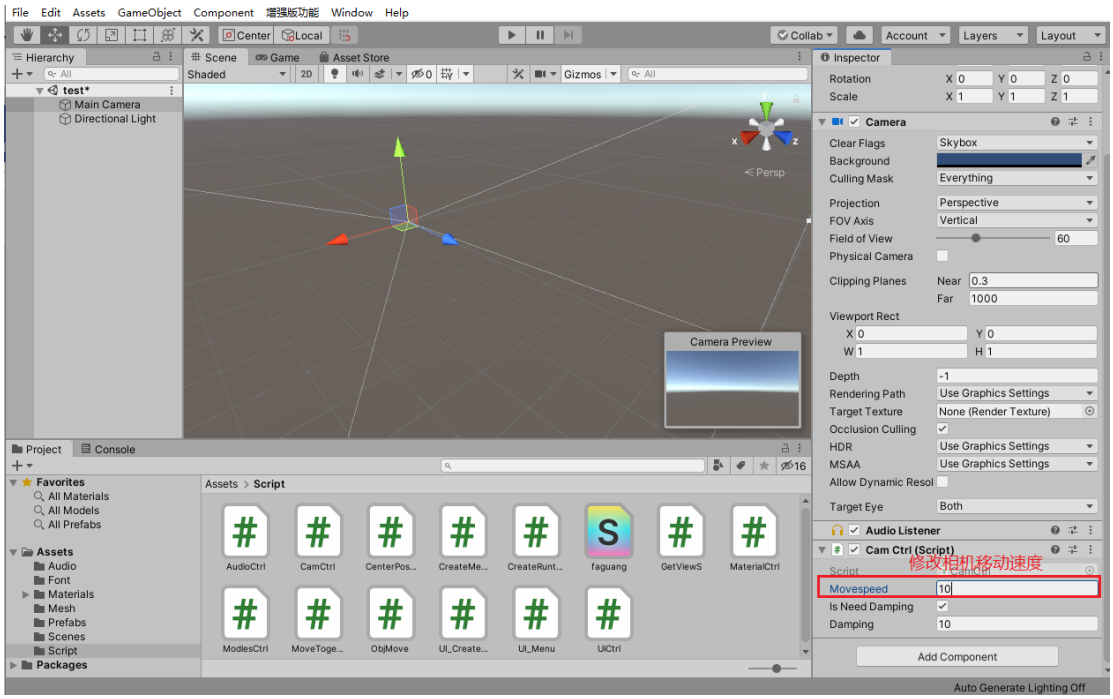
方法②：点击 Hierarchy 下的 Main Camera 对象，可以在 Inspector 面板看到 Main Camera 各个属性和组件，将 CamCtrl 脚本拖动至 Inspector 面板上。



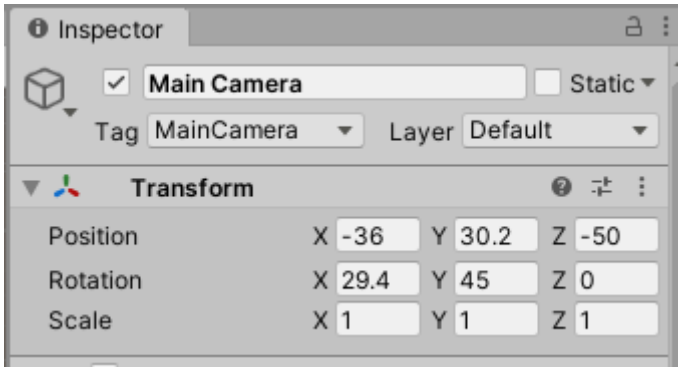
方法③：点击 Hierarchy 下的 Main Camera 对象，在 Inspector 面板下方找到 AddComponent 按钮，在搜索框中输入 CamCtrl，找到后点击即可。



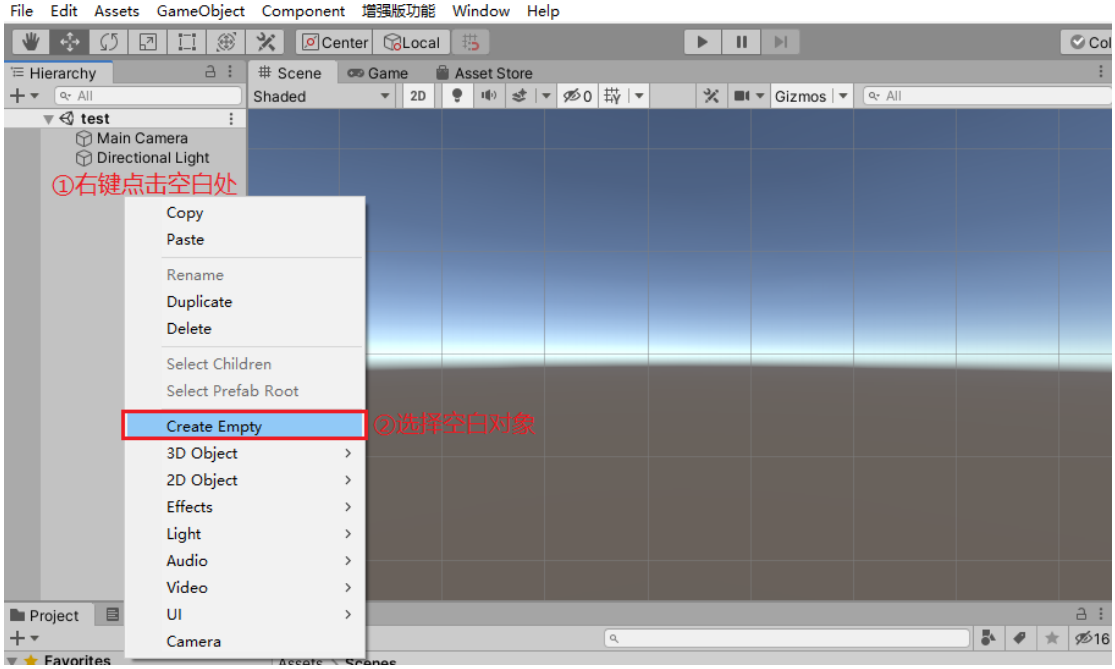
添加完脚本后，在脚本中将摄像机的移动速度 Movespeed 改成 10。

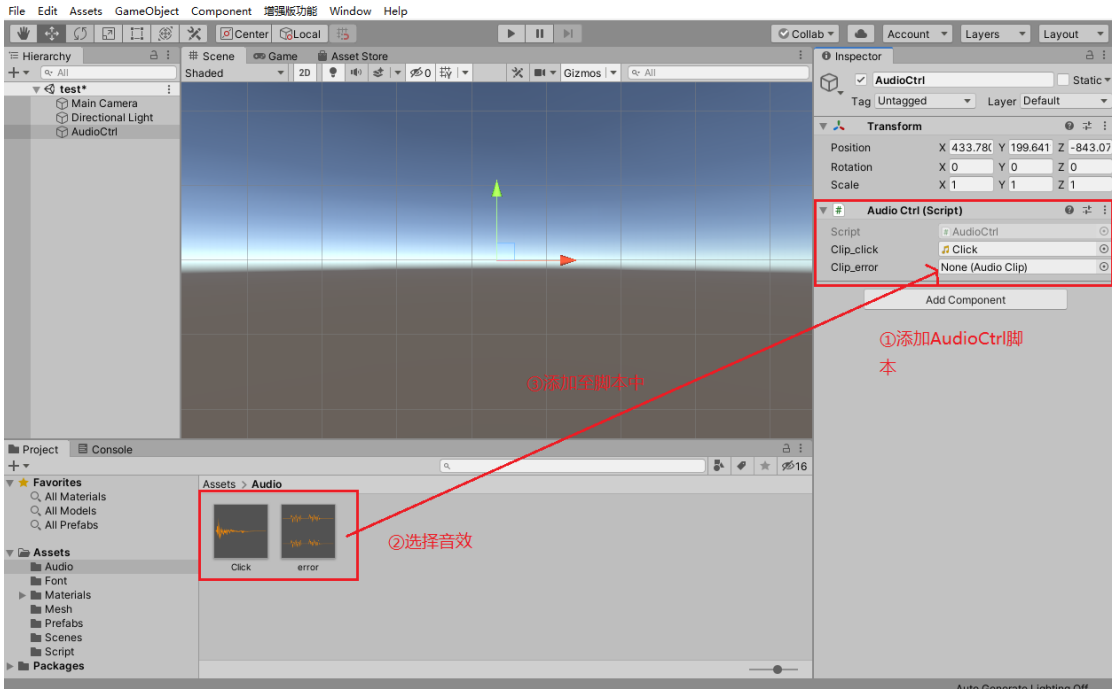


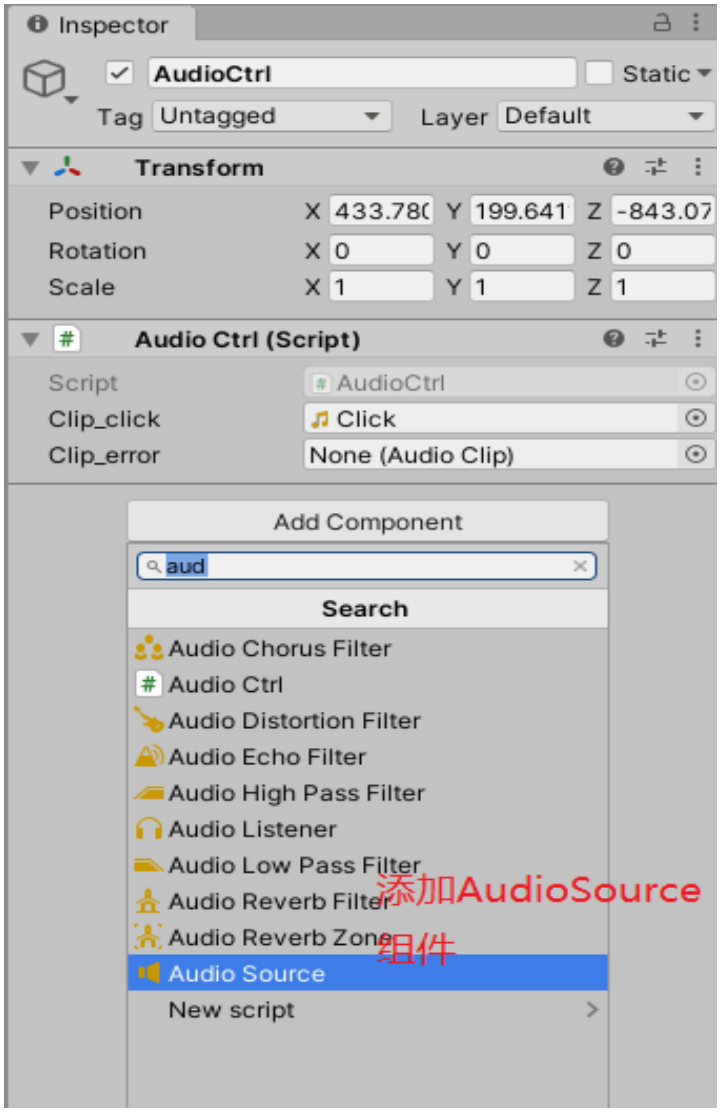
修改摄像机的 transform。



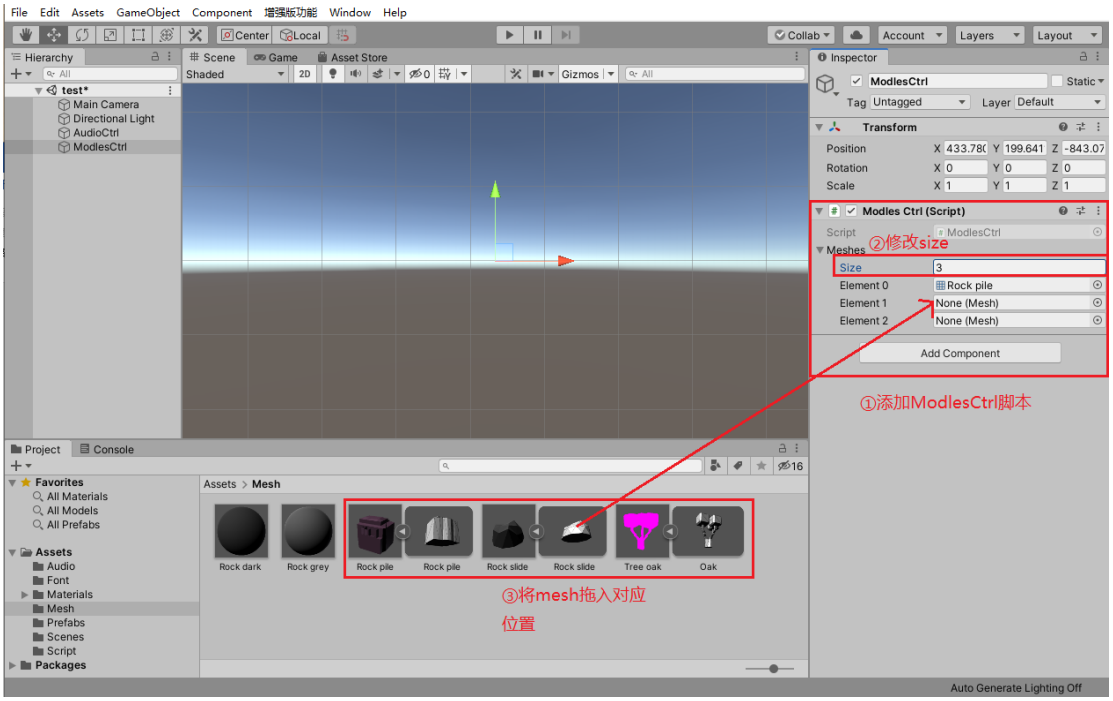
5. 添加音效控制器。在 Hierarchy 下的空白处右键点击，选择生成一个空白对象，修改对象名为 AudioCtrl，并为其添加脚本 AudoCtrl，并将 Project 面板中的 Audio 文件夹下的两个音效添加在刚才挂载的脚本上；在 Inspectors 面板中以第四步方法③的方式为其添加 AudioSource 组件。



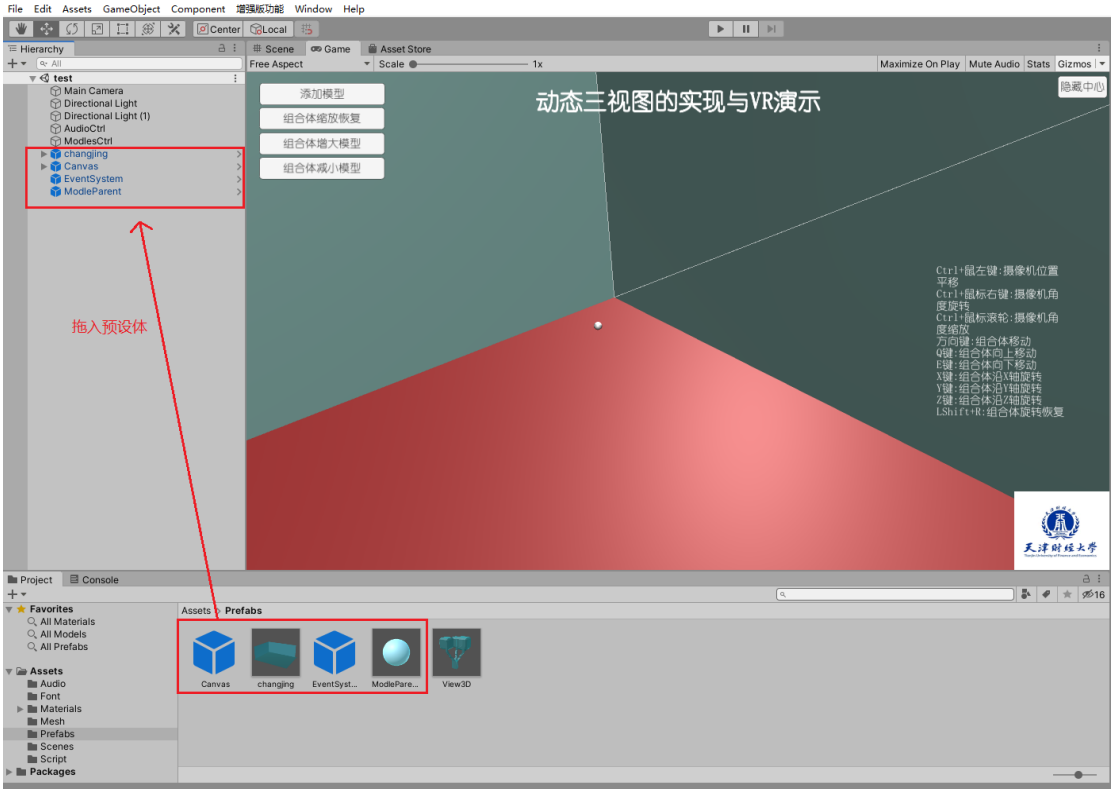

File Edit Assets GameObject Component 增强版功能 Window Help
Hierarchy
test*
Main Camera
Directional Light
AudioCtrl
Scene
Game
Asset Store
Inspector
AudioCtrl
Static
Tag Untagged
Layer Default
Transform
Position
Rotation
Scale
Audio Ctrl (Script)
Script
Clip_click
Clip_error
None (Audio Clip)
Add Component
①添加AudioCtrl脚本
③添加至脚本中
Project
Console
Favorites
All Materials
All Models
All Prefabs
Assets
Audio
Font
Materials
Mesh
Prefabs
Scenes
Script
Packages
Assets > Audio
Click
error
②选择音效


Inspector
AudioCtrl
Static
Tag Untagged
Layer Default
Transform
Position X 433.780 Y 199.641 Z -843.07
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Audio Ctrl (Script)
Script AudioCtrl
Clip_click Click
Clip_error None (Audio Clip)
Add Component
aud
Search
Audio Chorus Filter
Audio Ctrl
Audio Distortion Filter
Audio Echo Filter
Audio High Pass Filter
Audio Listener
Audio Low Pass Filter
Audio Reverb Filter
Audio Reverb Zone
Audio Source
New script
添加AudioSource组件

6. 以同样方法生成一个空对象，修改对象名为 ModlesCtrl，并为其添加脚本 ModlesCtrl。将脚本中 Meshes 的 size 修改为 3，并将 Project 面板下 Mesh 文件夹的三个模型拖入至脚本中对应位置

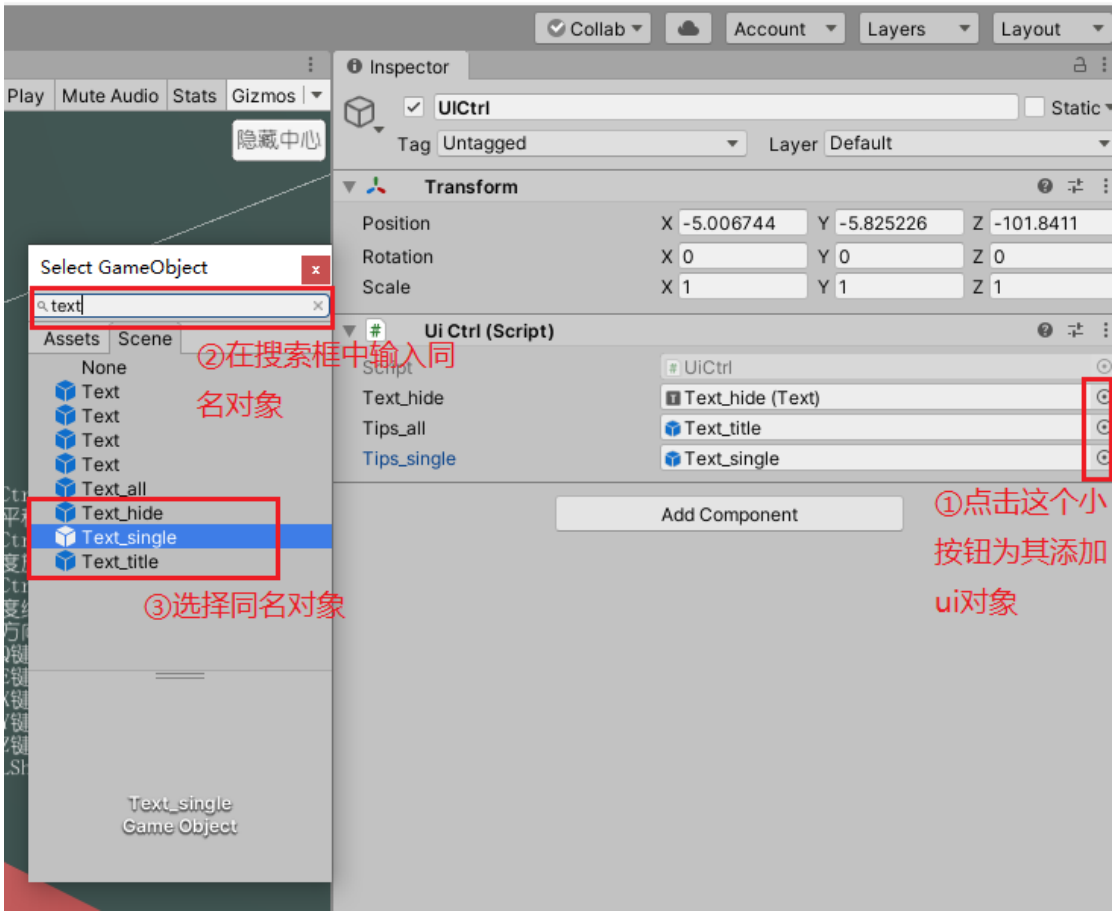


7. 将 Project 面板下 Prefab 文件夹下的四个预设体拖入到 Hierarchy 面板下，分别是 Canvas、场景、EventSystem 和 ModleParent。



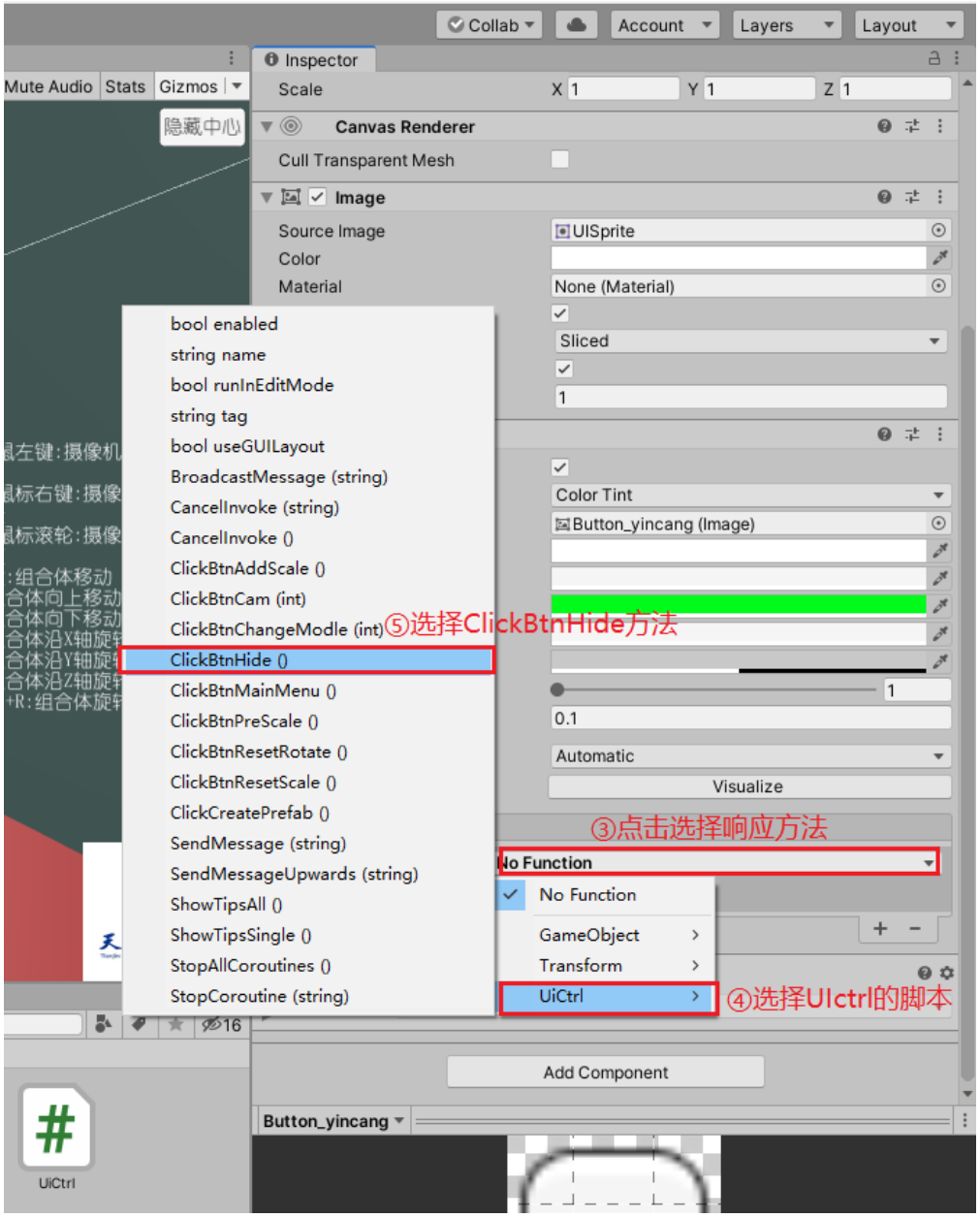
如果此处场景的光线过暗，则点击 Hierarchy 面板中的 Directional Light 对象，按下 Ctrl+D 键，将此对象复制，并将新产生的同名对象的 transform 组件中的 y 轴旋转值增加 180 度。

8. 在 Hierarchy 面板下添加空白对象 UICtrl，并给其添加脚本 UICtrl，在其 inspector
面板中为其三个属性添加 ui 对象。



点击 Hierarchy 面板中得 Canvas 对象，并找到子对象 Panel_btns，再找其子对象 Button_yincang，并再 inspector 面板中找到 Button 组件，找到 Onclick 事件，为事件添加响应方法，将创建好的 UICtrl 对象拖入到对应位置，选择 ClickBtnHide 方法并添加。


动态三视图的实现与VR演示
①选择Button对象
②将UICtrl拖拽入
Button_yincang


Collab
Account
Layers
Layout
Inspector
Mute Audio
Stats
Gizmos
隐藏中心
Scale
X 1
Y 1
Z 1
Canvas Renderer
Cull Transparent Mesh
Image
Source Image
UISprite
Color
Material
None (Material)
Sliced
1
Color Tint
Button_yincang (Image)
0.1
Automatic
Visualize
bool enabled
string name
bool runInEditMode
string tag
bool useGUILayout
BroadcastMessage (string)
CancelInvoke (string)
CancelInvoke ()
ClickBtnAddScale ()
ClickBtnCam (int)
ClickBtnChangeModle (int)
⑤选择ClickBtnHide方法
ClickBtnHide ()
ClickBtnMainMenu ()
ClickBtnPreScale ()
ClickBtnResetRotate ()
ClickBtnResetScale ()
ClickCreatePrefab ()
SendMessage (string)
SendMessageUpwards (string)
ShowTipsAll ()
ShowTipsSingle ()
StopAllCoroutines ()
StopCoroutine (string)
③点击选择响应方法
No Function
No Function
GameObject
Transform
UiCtrl
④选择UIctrl的脚本
Add Component
Button_yincang
UiCtrl

以同样的方法向其他四个按钮添加响应方法。Button_resetscale 选择方法 ClickBtnResetScale，Button_resetaddscale 选择方法 ClickBtnAddScale，Button_resetprescale 选择方法 ClickBtnPreScale，Button_Create 选择方法 ClickCreatePrefab。

9. 在 Hierarchy 面板中添加空白对象命名为 CreateMesh，并为其挂载脚本 CreateMesh。在 project 面板下打开 Script 文件夹，并双击打开 CreateMesh 脚本。在此脚本中，学生需使用顶点和索引数组制作一个正方体，以及自我创作一个复杂多面体，下面以创建一个正方体为例。

```
void Start()
{
    //vertices(顶点、必须)://放在start方法里在第一帧开始执行
    Vector3 Point = new Vector3(0, 0, 0);
    float length = 2; float width = 2; float heigth = 2;
    int vertices_count = 4 * 6;  //顶点数（每个面4个点，六个面）
    Vector3[] vertices = new Vector3[vertices_count];
    //前面的左下角的点
    vertices[0] = new Vector3(Point.x - length / 2, Point.y - heigth / 2, Point.z - width / 2);
    //前面的左上角的点
    vertices[1] = new Vector3(Point.x - length / 2, Point.y + heigth / 2, Point.z - width / 2);
    //前面的右下角的点
    vertices[2] = new Vector3(Point.x + length / 2, Point.y - heigth / 2, Point.z - width / 2);
    //前面的右上角的点
    vertices[3] = new Vector3(Point.x + length / 2, Point.y + heigth / 2, Point.z - width / 2);
    //后面的右下角的点
    vertices[4] = new Vector3(Point.x + length / 2, Point.y - heigth / 2, Point.z + width / 2);
    //后面的右上角的点
    vertices[5] = new Vector3(Point.x + length / 2, Point.y + heigth / 2, Point.z + width / 2);
    //后面的左下角的点
    vertices[6] = new Vector3(Point.x - length / 2, Point.y - heigth / 2, Point.z + width / 2);
    //后面的左上角的点
    vertices[7] = new Vector3(Point.x - length / 2, Point.y + heigth / 2, Point.z + width / 2);

    vertices[8] = vertices[6];                              //左
    vertices[9] = vertices[7];
    vertices[10] = vertices[0];
    vertices[11] = vertices[1];

    vertices[12] = vertices[2];                              //右
    vertices[13] = vertices[3];
    vertices[14] = vertices[4];
    vertices[15] = vertices[5];

    vertices[16] = vertices[1];                              //上
    vertices[17] = vertices[7];
    vertices[18] = vertices[3];
    vertices[19] = vertices[5];

    vertices[20] = vertices[2];                              //下
    vertices[21] = vertices[4];
    vertices[22] = vertices[0];
    vertices[23] = vertices[6];
```

```
CreateMesh
    //triangles(索引三角形、必须):unity顺时针渲染
    int SplitTriangle = 6 * 2;//分割三角形数量
    int triangles_cout = SplitTriangle * 3;                    //索引三角形的索引点个数
    int[] triangles = new int[triangles_cout];                //索引三角形数组
    for (int i = 0, vi = 0; i < triangles_cout; i += 6, vi += 4)
    {
        triangles[i] = vi;
        triangles[i + 1] = vi + 1;
        triangles[i + 2] = vi + 2;

        triangles[i + 3] = vi + 3;
        triangles[i + 4] = vi + 2;
        triangles[i + 5] = vi + 1;

    }
    //负载属性与mesh
    Mesh mesh = new Mesh();
    mesh.vertices = vertices;
    mesh.triangles = triangles;
    mesh.RecalculateBounds();
    mesh.RecalculateNormals();
    StartCoroutine(Create_mesh(mesh));
}

1 个引用
IEnumerator Create_mesh(Mesh mesh)
{
    //使用协程防止创建不成功
    yield return new WaitForSeconds(1.0f);
    ModlesCtrl.instanse.AddMesh(mesh);//调用addmesh方法添加构建好的mesh

}
```

10. 打开 GetViewS 脚本，编写投影计算方法。

```
public class GetViewS : MonoBehaviour
{
    private Mesh mesh_top, mesh_right, mesh_front;
    private Mesh modle_mesh;
    // Start is called before the first frame update
    0 个引用
    void Start()
    {
        modle_mesh = transform.GetComponent<ObjMove>().main_modle.GetComponent<MeshFilter>().mesh;
        mesh_front = new Mesh();
        mesh_right = new Mesh();
        mesh_top = new Mesh();

    }


    0 个引用
    void Update()
    {
        //每一帧将当前模型的mesh的顶点和索引赋值给三个方向投影对象
        modle_mesh = transform.GetComponent<ObjMove>().GetModleMesh;
        mesh_front.vertices = transform.GetComponent<ObjMove>().GetVeiticesPos;
        mesh_right.vertices = transform.GetComponent<ObjMove>().GetVeiticesPos;
        mesh_top.vertices = transform.GetComponent<ObjMove>().GetVeiticesPos;
        mesh_front.triangles = modle_mesh.triangles;
        mesh_right.triangles = modle_mesh.triangles;
        mesh_top.triangles = modle_mesh.triangles;
        TouyingOnFront();
    }
    1 个引用
    void TouyingOnFront()
    {


        int len = mesh_front.vertices.Length;//顶点数组长度
        Vector3[] vertices = mesh_front.vertices;
        for(int i=0;i<len;++i)
        {//正投影
            vertices[i] = transform.InverseTransformPoint(vertices[i]);
            //正投影变换矩阵
            int[,] matrix_ = { { 1, 0, 0 },
                               { 0, 1, 0 },
```

```
                    { 0, 1, 0 },
                    { 0, 0, 0 } };
        CalculateMatrix(ref vertices[i], matrix_);
    }
    mesh_front.vertices = vertices;
    vertices = mesh_right.vertices;
    for (int i = 0; i < len; ++i)
    {//侧投影
        vertices[i] = transform.InverseTransformPoint(vertices[i]);
        //侧投影变换矩阵
        int[,] matrix_ = { { 0, 0, 0 },
                    { 0, 1, 0 },
                    { 0, 0, 1 } };
        //vertices[i].x = 0;
        CalculateMatrix(ref vertices[i], matrix_);
    }
    mesh_right.vertices = vertices;
    vertices = mesh_top.vertices;
    for (int i = 0; i < len; ++i)
    { //俯视
        vertices[i] = transform.InverseTransformPoint(vertices[i]);
        int[,] matrix_ = { { 1, 0, 0 },
                    { 0, 0, 0 },
                    { 0, 0, 1 } };
        CalculateMatrix(ref vertices[i], matrix_);
    }
    mesh_top.vertices = vertices;
    //将计算后的投影分别赋值给投影对象
    transform.GetComponent<ObjMove>().view_front.GetComponent<MeshFilter>().mesh = mesh_front;
    transform.GetComponent<ObjMove>().view_right.GetComponent<MeshFilter>().mesh = mesh_right;
    transform.GetComponent<ObjMove>().view_top.GetComponent<MeshFilter>().mesh = mesh_top;

}
3 个引用
void CalculateMatrix(ref Vector3 pos,int[,] matrix)
{
    //投影计算方法
    //此处没有进行矩阵安全检查,一定小心输入问题
    float temp_x = pos[0] * matrix[0, 0] + pos[0] * matrix[0, 1] + pos[0] * matrix[0, 2];
    float temp_y = pos[1] * matrix[1, 0] + pos[1] * matrix[1, 1] + pos[1] * matrix[1, 2];
    float temp_z = pos[2] * matrix[2, 0] + pos[2] * matrix[2, 1] + pos[2] * matrix[2, 2];
    pos = new Vector3(temp_x, temp_y, temp_z);
}
```

11. 检查错误，完成后在 unity 界面点击开始按钮，并按照网页版实验（一）步骤进行观察实验。