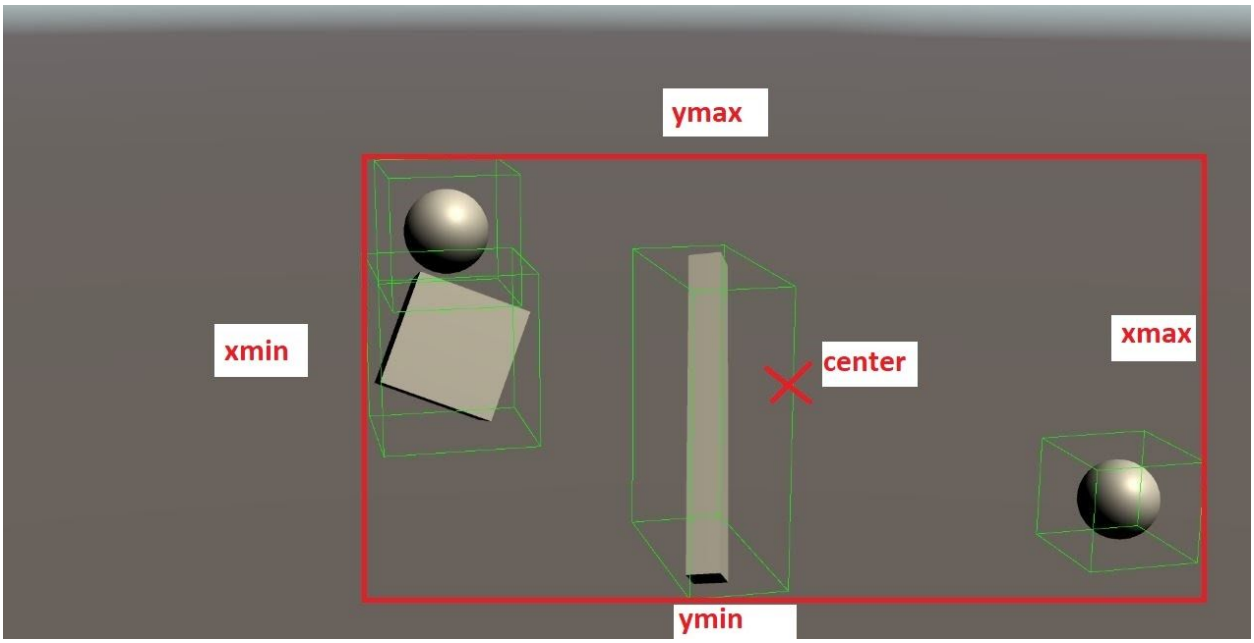
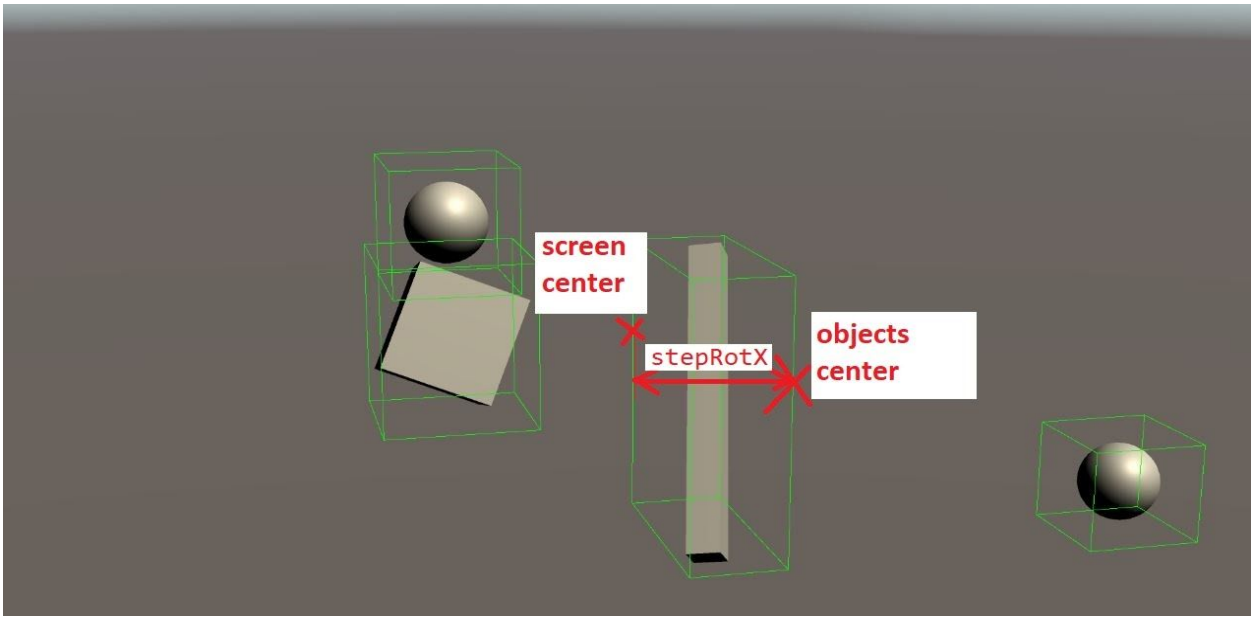
Описание решения

1) Раз нужно учитывать, как объекты проецируются на экран, то сразу проецируем их на экран (используем 8 вершин из bounding box каждого объекта). В дальнейшем работаем только в screen space.

2) Определяем габариты всех объектов в совокупности и их общий центр.

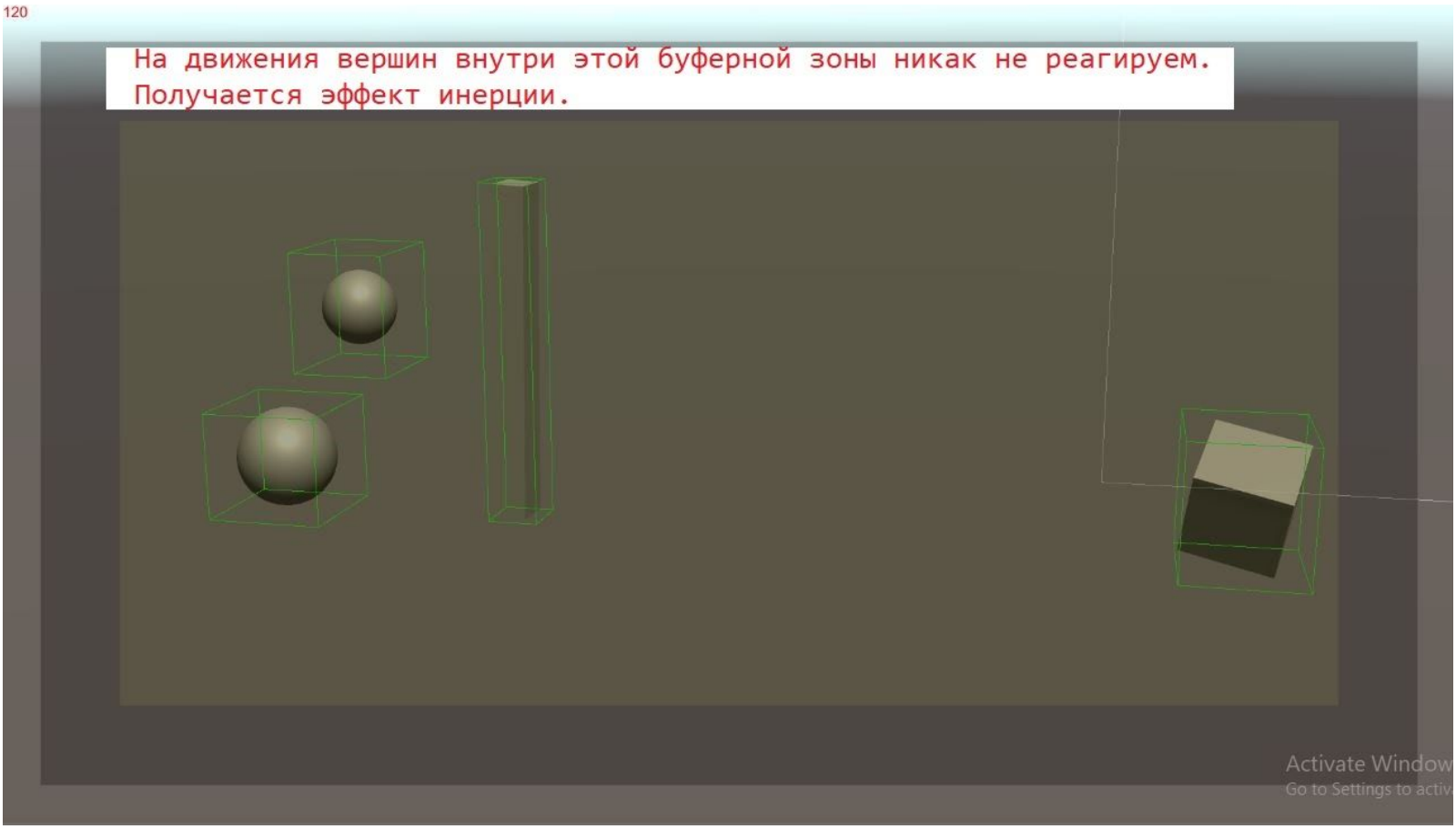


3) Определяем угол, на который надо повернуть (stepRotX, stepRotY):
Для этого сравниваем центр объектов с центром экрана. Чем дальше они друг от друга, тем сильнее вращаем.

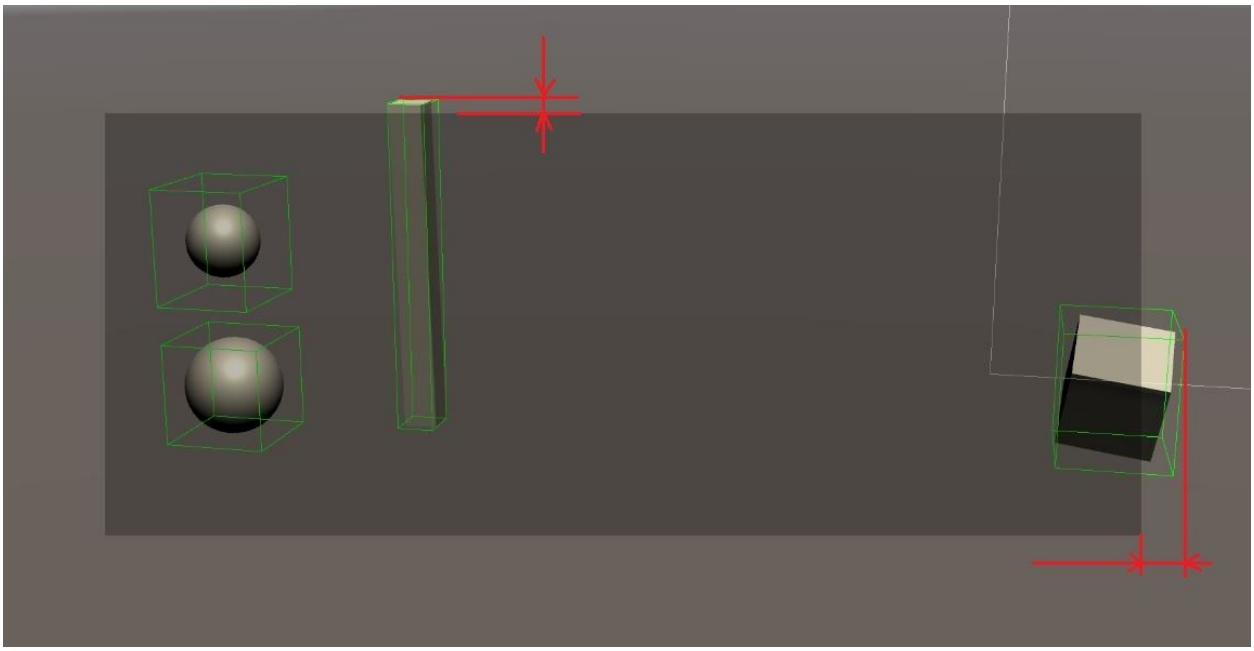


4) вращаем:
cam.transform.Rotate(Vector3.up * stepRotX, Space.World);
cam.transform.Rotate(Camera.main.transform.right * stepRotY, Space.World);

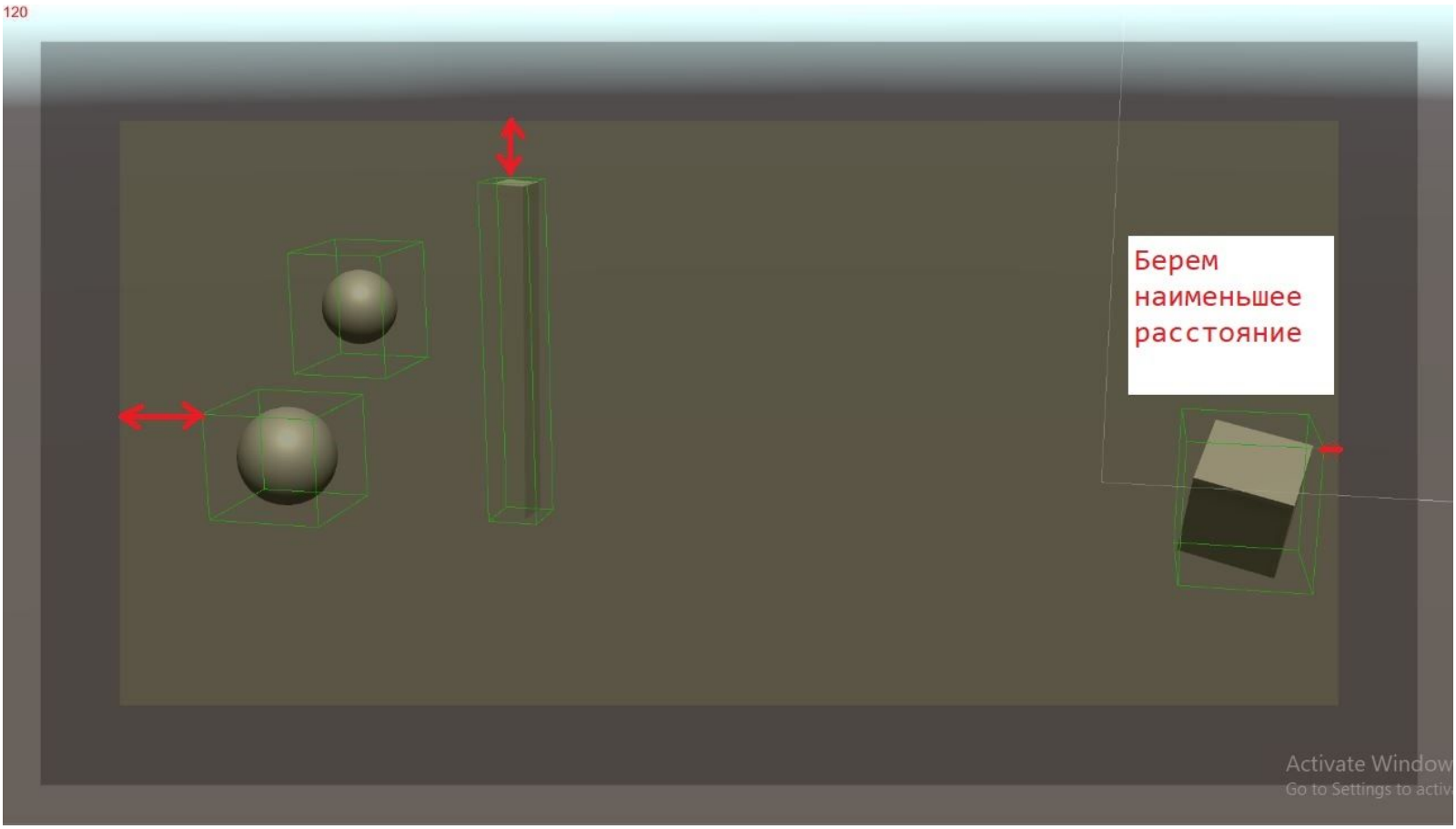
5) добавляем область на экране, в которую должны быть вписаны объекты (PaddingArea)
Для инерции - добавляем вторую область внутри первой (ThresholdArea).
На движения между областями никак не реагируем.



6) Вычисляем угол, на который надо изменить FOV
а) Если есть вершины вне разрешенных областей - берем расстояние до самой отдаленной.
И увеличиваем FOV на это значение:
float stepFOV = stepFovMax * Time.deltaTime * speedFOV;
Camera.main.fieldOfView += stepFOV;



б) Если все вершины внутри наименьшей области - берем минимальной расстояние от границы, и на это значение уменьшаем FOV



7) Вывод:
В результате поворот и фокусировка работают плавно, ускоряясь в начале и замедляясь в конце.
Буферная зона создает эффект инерции и камера не реагирует на мелкие колебания объектов.
Иногда объекты вылетают за пределы области, но камера быстро подстраивается.
Если нужно полностью исключить вылеты, то надо внутри RandomCubeMoves.cs при задании новых координат сразу поворачивать и фокусировать камеру в том же кадре.

Все необходимые объекты для работы в solution.unitypackage.
Главный скрипт - CameraLook.cs (есть настройки скорости и отступов).
Остальные скрипты ему помогают.
Видео пример - https://youtu.be/EtimzzM8XHA