摄像机平滑跟随--方法一

using UnityEngine;

using System.Collections;

public class SmoothFollowerObj : MonoBehaviour {

private Vector3 targetPosition;

private Vector3 position;

private Vector3 velocity;

private float smoothingTime;

private float prediction;

public SmoothFollowerObj(float smoothingTime)

{

targetPosition = Vector3.zero;

position = Vector3.zero;

velocity = Vector3.zero;

this.smoothingTime = smoothingTime;

prediction = 1;

}

public SmoothFollowerObj(float smoothingTime, float prediction)

{

targetPosition = Vector3.zero;

position = Vector3.zero;

velocity = Vector3.zero;

this.smoothingTime = smoothingTime;

this.prediction = prediction;

}

// Update should be called once per frame

public Vector3 Update(Vector3 targetPositionNew, float deltaTime)

{

Vector3 targetVelocity = (targetPositionNew - targetPosition) / deltaTime;

targetPosition = targetPositionNew;

float d = Mathf.Min(1, deltaTime / smoothingTime);

velocity = velocity \* (1 - d) + (targetPosition + targetVelocity \* prediction - position) \* d;

position += velocity \* Time.deltaTime;

return position;

}

public Vector3 Update(Vector3 targetPositionNew, float deltaTime, bool reset)

{

if (reset)

{

targetPosition = targetPositionNew;

position = targetPositionNew;

velocity = Vector3.zero;

return position;

}

return Update(targetPositionNew, deltaTime);

}

public Vector3 GetPosition() { return position; }

public Vector3 GetVelocity() { return velocity; }

}

下面是摄像机的代码

using UnityEngine;

using System.Collections;

public class NoGravityCamera : MonoBehaviour {

public GameObject character;

public Vector3 positionVector;

public Vector3 lookVector;

private SmoothFollowerObj posFollow;

private SmoothFollowerObj lookFollow;

private Vector3 lastVelocityDir;

private Vector3 lastPos;

// Use this for initialization

void Start()

{

//positionVector = new Vector3(0, 2, 4);

//lookVector = new Vector3(0, 0, 1.5f);

posFollow = new SmoothFollowerObj(0.5f, 0.5f);

lookFollow = new SmoothFollowerObj(0.1f, 0.0f);

posFollow.Update(transform.position, 0, true);

lookFollow.Update(character.transform.position, 0, true);

lastVelocityDir = character.transform.forward;

lastPos = character.transform.position;

}

// Update is called once per frame

void LateUpdate()

{

lastVelocityDir += (character.transform.position - lastPos) \* 8;

lastPos = character.transform.position;

lastVelocityDir += character.transform.forward \* Time.deltaTime;

lastVelocityDir = lastVelocityDir.normalized;

Vector3 horizontal = transform.position - character.transform.position;

Vector3 horizontal2 = horizontal;

Vector3 vertical = character.transform.up;

Vector3.OrthoNormalize(ref vertical, ref horizontal2);

if (horizontal.sqrMagnitude > horizontal2.sqrMagnitude) horizontal = horizontal2;

transform.position = posFollow.Update(

character.transform.position + horizontal \* Mathf.Abs(positionVector.z) + vertical \* positionVector.y,

Time.deltaTime

);

horizontal = lastVelocityDir;

Vector3 look = lookFollow.Update(character.transform.position + horizontal \* lookVector.z - vertical \* lookVector.y, Time.deltaTime);

transform.rotation = Quaternion.FromToRotation(transform.forward, look - transform.position) \* transform.rotation;

}

}

摄像机平滑过渡

#pragma strict

var cam01:GameObject;

var cam01Pos:GameObject;

var cam02Pos:GameObject;

// 用于判断摄像机是否在移动

var isMoving:boolean = false;

var dx:float = 0;

var dy:float = 0;

var dz:float = 0;

function Start ()

{

// 初始化摄像机

cam01 = GameObject.Find("Camera01");

// 初始化两个摄像机位置

cam01Pos = GameObject.Find("Cam01Pos");

cam02Pos = GameObject.Find("Cam02Pos");

// 默认情况下摄像机处于位置一

cam01.transform.position = cam01Pos.transform.position;

}

function Update ()

{

if(Input.GetKeyUp(KeyCode.Alpha1))

{

// 从位置二过渡到位置一

dx = cam01Pos.transform.position.x - cam01.transform.position.x;

dy = cam01Pos.transform.position.y - cam01.transform.position.y;

dz = cam01Pos.transform.position.z - cam01.transform.position.z;

dx /= 100.0f;

dy /= 100.0f;

dz /= 100.0f;

isMoving = true;

}

else if(Input.GetKeyUp(KeyCode.Alpha2))

{

// 从位置一过渡到位置二

dx = cam02Pos.transform.position.x - cam01.transform.position.x;

dy = cam02Pos.transform.position.y - cam01.transform.position.y;

dz = cam02Pos.transform.position.z - cam01.transform.position.z;

dx /= 100.0f;

dy /= 100.0f;

dz /= 100.0f;

isMoving = true;

}

// 摄像机过渡动画

if(isMoving == true)

{

cam01.transform.position.x += dx;

cam01.transform.position.y += dy;

cam01.transform.position.z += dz;

// 判断摄像机是否到达指定位置

if(cam01Pos.transform.position == cam01.transform.position)

{

isMoving = false;

}

else if(cam02Pos.transform.position == cam01.transform.position)

{

isMoving = false;

}

}

}

摄像机旋转位移跟随

public Transform target;

public float distance ;

public float targetHeight;

private float x = 0.0f;

private float y = 0.0f;

void Start () {

var angles = transform.eulerAngles;

x = angles.x;

y = angles.y;

}

void LateUpdate ()

{

if(!target)

return;

y = target.eulerAngles.y;

// ROTATE CAMERA:

Quaternion rotation = Quaternion.Euler(x, y, 0);

transform.rotation = rotation;

// POSITION CAMERA:

Vector3 position = target.position - (rotation \* Vector3.forward \* distance + new Vector3(0,-targetHeight,0));

transform.position = position;

}

摄像机跟随target快速位移和旋转‘

public Transform target;

public float distance ;

public float targetHeight;

public float PitchAngle;

private float x = 0.0f;

private float y = 0.0f;

void Start () {

var angles = transform.eulerAngles;

x = angles.x;

y = angles.y;

}

void LateUpdate ()

{

if(!target)

return;

y = target.eulerAngles.y;

// ROTATE CAMERA:

Quaternion rotation = Quaternion.Euler(x-PitchAngle, y, 0);

transform.rotation = rotation;

// POSITION CAMERA:

Vector3 position = target.position - (rotation \* Vector3.forward \* distance + new Vector3(0,-targetHeight,0));

transform.position = position;

}

摄像机平滑跟随--方法二

public Transform target = null;

public float height = 1f;

public float positionDamping = 3f;

public float velocityDamping = 3f;

public float distance = 4f;

public LayerMask ignoreLayers = -1;

private RaycastHit hit = new RaycastHit();

private Vector3 prevVelocity = Vector3.zero;

private LayerMask raycastLayers = -1;

private Vector3 currentVelocity = Vector3.zero;

void Start()

{

raycastLayers = ~ignoreLayers;

}

void FixedUpdate()

{

currentVelocity = Vector3.Lerp(prevVelocity, target.root.rigidbody.velocity, velocityDamping \* Time.deltaTime);

currentVelocity.y = 0;

prevVelocity = currentVelocity;

}

void LateUpdate()

{

float speedFactor = Mathf.Clamp01(target.root.rigidbody.velocity.magnitude / 70.0f);

camera.fieldOfView = Mathf.Lerp(55, 72, speedFactor);

float currentDistance = Mathf.Lerp(7.5f, 6.5f, speedFactor);

currentVelocity = currentVelocity.normalized;

Vector3 newTargetPosition = target.position + Vector3.up \* height;

Vector3 newPosition = newTargetPosition - (currentVelocity \* currentDistance);

newPosition.y = newTargetPosition.y;

Vector3 targetDirection = newPosition - newTargetPosition;

if(Physics.Raycast(newTargetPosition, targetDirection, out hit, currentDistance, raycastLayers))

newPosition = hit.point;

transform.position = newPosition;

transform.LookAt(newTargetPosition);

}

Camare shake

var originPosition:Vector3;

var originRotation:Quaternion;

var shake\_decay: float;

var shake\_intensity: float;;

function OnGUI () {

if (GUI.Button (Rect (20,40,80,20), "Shake")) {

Shake();

}

}

function Update(){

if(shake\_intensity > 0){

transform.position = originPosition + Random.insideUnitSphere \* shake\_intensity;

transform.rotation = Quaternion(

originRotation.x + Random.Range(-shake\_intensity,shake\_intensity)\*.2,

originRotation.y + Random.Range(-shake\_intensity,shake\_intensity)\*.2,

originRotation.z + Random.Range(-shake\_intensity,shake\_intensity)\*.2,

originRotation.w + Random.Range(-shake\_intensity,shake\_intensity)\*.2);

shake\_intensity -= shake\_decay;

}

}

function Shake(){

originPosition = transform.position;

originRotation = transform.rotation;

shake\_intensity = .3;

shake\_decay = 0.002;

}

摄像机平滑跟随代码

using UnityEngine;

using System.Collections;

public class SmoothFollowerObj {

private Vector3 targetPosition;

private Vector3 position;

private Vector3 velocity;

private float smoothingTime;

private float prediction;

public SmoothFollowerObj(float smoothingTime) {

targetPosition = Vector3.zero;

position = Vector3.zero;

velocity = Vector3.zero;

this.smoothingTime = smoothingTime;

prediction = 1;

}

public SmoothFollowerObj(float smoothingTime, float prediction) {

targetPosition = Vector3.zero;

position = Vector3.zero;

velocity = Vector3.zero;

this.smoothingTime = smoothingTime;

this.prediction = prediction;

}

// Update should be called once per frame

public Vector3 Update(Vector3 targetPositionNew, float deltaTime) {

Vector3 targetVelocity = (targetPositionNew-targetPosition)/deltaTime;

targetPosition = targetPositionNew;

float d = Mathf.Min(1,deltaTime/smoothingTime);

velocity = velocity\*(1-d) + (targetPosition+targetVelocity\*prediction-position)\*d;

position += velocity\*Time.deltaTime;

return position;

}

public Vector3 Update(Vector3 targetPositionNew, float deltaTime, bool reset) {

if (reset) {

targetPosition = targetPositionNew;

position = targetPositionNew;

velocity = Vector3.zero;

return position;

}

return Update(targetPositionNew, deltaTime);

}

public Vector3 GetPosition() { return position; }

public Vector3 GetVelocity() { return velocity; }

}

public class CameraController : MonoBehaviour {

public GameObject character;

public Vector3 positionVector;

public Vector3 lookVector;

private SmoothFollowerObj posFollow;

private SmoothFollowerObj lookFollow;

private Vector3 lastVelocityDir;

private Vector3 lastPos;

// Use this for initialization

void Start () {

positionVector=new Vector3(0,2,4);

lookVector=new Vector3(0,0,1.5f);

posFollow = new SmoothFollowerObj(0.5f,0.5f);

lookFollow = new SmoothFollowerObj(0.1f,0.0f);

posFollow.Update(transform.position,0,true);

lookFollow.Update(character.transform.position,0,true);

lastVelocityDir = character.transform.forward;

lastPos = character.transform.position;

}

// Update is called once per frame

void LateUpdate () {

lastVelocityDir += (character.transform.position-lastPos)\*8;

lastPos = character.transform.position;

lastVelocityDir += character.transform.forward\*Time.deltaTime;

lastVelocityDir = lastVelocityDir.normalized;

Vector3 horizontal = transform.position-character.transform.position;

Vector3 horizontal2 = horizontal;

Vector3 vertical = character.transform.up;

Vector3.OrthoNormalize(ref vertical,ref horizontal2);

if (horizontal.sqrMagnitude > horizontal2.sqrMagnitude) horizontal = horizontal2;

transform.position = posFollow.Update(

character.transform.position + horizontal\*Mathf.Abs(positionVector.z) + vertical\*positionVector.y,

Time.deltaTime

);

horizontal = lastVelocityDir;

Vector3 look = lookFollow.Update(character.transform.position + horizontal\*lookVector.z - vertical\*lookVector.y, Time.deltaTime);

transform.rotation = Quaternion.FromToRotation(transform.forward, look-transform.position) \* transform.rotation;

}

}

Unity3D 摄像机平滑跟随角色

摄像机跟随主角移动有多种方式，第一人称视角的话摄像机保持和主角一直的位置和朝向，并增加一些平滑移动即可。第三人称的（包括越肩和斜45读视角），就复杂一些，因为有时候会导致摄像机和主角之间有东西挡着，照不到主角。

这种情况可以用射线来判断和主角之间是否有遮挡物，并设置多个位置，便利寻找不会被遮挡的位置。

官方实例

void Awake() {

player = GameObject.FindGameObjectWithTag(Tags.player).transform;

offset = transform.position - player.position;

offset = new Vector3(0, offset.y, offset.z);

}

void UpdateCamePos () {

Vector3 beginPos = player.position + offset;

Vector3 endPos = player.position + offset.magnitude \* Vector3.up;

Vector3 pos1 = Vector3.Lerp(beginPos, endPos, 0.25f);

Vector3 pos2 = Vector3.Lerp(beginPos, endPos, 0.5f);

Vector3 pos3 = Vector3.Lerp(beginPos, endPos, 0.75f);

Vector3[] posArray = new Vector3[] { beginPos, pos1, pos2, pos3, endPos };

Vector3 targetPos = posArray[0];

for (int i = 0; i < 5; i++) {

RaycastHit hitinfo;

if (Physics.Raycast(posArray[i], player.position - posArray[i], out hitinfo)) {

if (hitinfo.collider.tag != Tags.player) {

continue;

} else {

targetPos = posArray[i];

break;

}

} else {

targetPos = posArray[i];

break;

}

}

这个函数在Update()中调用，可以更新摄像机的位置。这里的offset 即使初始情况下摄像机和主角的位置差，是在编辑场景的时候调整的。

Void smoothLookAt()

{

Vector3 relPlayerPosition = player.position - transform.position;

Quaternion lookAtRotation = Quaternion.LookRotation(relPlayerPosition ,Vector3.up);

Transform.rotation = Quaternion.Lerp(transform.rotation,lookAtRotation,smooth\*Time.deltaTime);

}

这是平滑移动摄像机朝向，对准主角的函数，同样在Update()中调用。

第三人称越肩视角和上面的方法类似，不过endPos 应该是离主角更近的位置而不是取主角的正上方，效果就是在一些射击游戏中，当玩家背靠墙的时候摄像机会离主角很近，而不会在墙的后面。

自己实现的A\*算法

以前一直用的是漏斗平滑算法，但对于八格子寻路来说，效果其实不是很好，然后coc类游戏的话，就需要更加平滑的路线。于是自己想了一套方案，顺便发现了以前判断直线穿越格子的函数的问题。

先矫正以前的函数：

public bool CheckCanGoForward(Vector3 startPos, Vector3 endPos, int extNum)

{

//init: gather starting location information.

int startCell = GetCellIndex(startPos);

int lastCell = GetCellIndex (endPos);

if(lastCell < 0)

{

Debug.Log("endPos is valid");

return false;

}

float maxDis = (endPos - startPos).sqrMagnitude;

Vector2 dir = new Vector2 (endPos.x - startPos.x, endPos.z - startPos.z);

bool bInBounds = ( startCell >= 0 && startCell < NumberOfCells );

if (!bInBounds)

{

System.Diagnostics.Debug.Assert(bInBounds, "starting position of the ray is not in bounds in call to Raycast2D" +

"Add logic to find the starting cell when the ray position starts out of bounds");

return false;

}

int X = GetColumn(startCell);

int Y = GetRow(startCell);

int stepX = Math.Sign(dir.x);

int stepY = Math.Sign(dir.y);

float nearestGridX = (stepX < 0) ? (startPos.x) : (startPos.x + m\_cellSize);

float nearestGridY = (stepY < 0) ? (startPos.z - m\_cellSize) : (startPos.z);

float thetaInDegrees = Vector3.Angle( kXAxis, dir);

float thetaInRadians = thetaInDegrees \* Mathf.Deg2Rad;

float cosTheta = Mathf.Cos( thetaInRadians );

float sinTheta = Mathf.Sin( thetaInRadians );

//parametric form requires taking ray.Position as the origin, hence the " - ray.Position"

float tMaxX = Math.Abs((nearestGridX - startPos.x) / cosTheta);

float tMaxY = Math.Abs((nearestGridY - startPos.z) / sinTheta);

float tDeltaX = Math.Abs(m\_cellSize / cosTheta);

float tDeltaY = Math.Abs(m\_cellSize / sinTheta);

//loop: traverse the cells until there is a collision or ray is out of bounds.

bool bHitMapEdge = false;

int maxCount = 100;

int nowCount = 0;

while (nowCount < maxCount)

{

nowCount++;

Vector3 nowPos;

float xMax = tMaxX + tDeltaX;

float yMax = tMaxY + tDeltaY;

if (xMax < yMax)

{

tMaxX += tDeltaX;

X += stepX;

nowPos = startPos + new Vector3(tMaxX \* cosTheta, 0f, tMaxX \* sinTheta);

}

else

{

tMaxY += tDeltaY;

Y += stepY;

nowPos = startPos + new Vector3(tMaxY \* cosTheta, 0f, tMaxY \* sinTheta);

}

int cell = GetCellIndex(nowPos);

if(!checkCanGo(cell, extNum, null))

{

return false;

}

if((nowPos - startPos).sqrMagnitude >= maxDis)

{

return true;

}

}

Debug.Log ("can not find forward pos to Go");

return false;

}

这个就不多说了，重点讲下平滑算法：

1.第一步是减少点的数量，加快计算速度。那么哪些点可以不要？就是一条线段，由四个点组成，那么我们只取端点，中间两点可以省略。

List<Vector3> newPathList = new List<Vector3> ();

Vector3 preDir = Vector3.zero;

for(int i = path.Length - 1; i >= 1; i--)

{

Vector3 nowDir = path[i] - path[i - 1];

if(nowDir == preDir)

{

continue;

}

else

{

preDir = nowDir;

newPathList.Add(path[i]);

}

}

然后，对于一个点，我们需要判断它是否可以跳过下一个点而到达下下个点，如果可以，那么就移除中间点。

for(int i = newPathList.Count - 1; i >= 2; i--)

{

if(grid.CheckCanGoForward(newPathList[i], newPathList[i - 2], extNum))

{

newPathList.RemoveAt(i - 1);

}

}

就这么简单，但效果却非常好，我看了下市面上的其他游戏的源代码，大部分都是采用这种方式，看来应该可以采用。

A\*寻路

1.怎么判断直线通往目标的路径上有障碍物？

这种情况下，DDA算法比A\*更快，如果没有障碍物，那么直接前进即可，不用A\*算法，对于coc这种百人寻路是可以节省不少性能的。

代码如下：

public bool CheckCanGoForward(Vector3 startPos, Vector3 endPos, int extNum)

{

//init: gather starting location information.

int startCell = GetCellIndex(startPos);

int lastCell = GetCellIndex (endPos);

if(lastCell < 0)

{

Debug.Log("endPos is valid");

return false;

}

float maxDis = (endPos - startPos).sqrMagnitude;

Vector2 dir = new Vector2 (endPos.x - startPos.x, endPos.z - startPos.z);

bool bInBounds = ( startCell >= 0 && startCell < NumberOfCells );

if (!bInBounds)

{

System.Diagnostics.Debug.Assert(bInBounds, "starting position of the ray is not in bounds in call to Raycast2D" +

"Add logic to find the starting cell when the ray position starts out of bounds");

return false;

}

int X = GetColumn(startCell);

int Y = GetRow(startCell);

int stepX = Math.Sign(dir.x);

int stepY = Math.Sign(dir.y);

float nearestGridX = (stepX < 0) ? (startPos.x) : (startPos.x + m\_cellSize);

float nearestGridY = (stepY < 0) ? (startPos.z - m\_cellSize) : (startPos.z);

float thetaInDegrees = Vector3.Angle( kXAxis, dir);

float thetaInRadians = thetaInDegrees \* Mathf.Deg2Rad;

float cosTheta = Mathf.Cos( thetaInRadians );

float sinTheta = Mathf.Sin( thetaInRadians );

//parametric form requires taking ray.Position as the origin, hence the " - ray.Position"

float tMaxX = Math.Abs((nearestGridX - startPos.x) / cosTheta);

float tMaxY = Math.Abs((nearestGridY - startPos.z) / sinTheta);

float tDeltaX = Math.Abs(m\_cellSize / cosTheta);

float tDeltaY = Math.Abs(m\_cellSize / sinTheta);

//loop: traverse the cells until there is a collision or ray is out of bounds.

bool bHitMapEdge = false;

int prevX = X;

int prevY = Y;

int maxCount = 100;

int nowCount = 0;

while (nowCount < maxCount)

{

nowCount++;

if (tMaxX < tMaxY)

{

prevX = X;

tMaxX += tDeltaX;

X += stepX;

}

else

{

prevY = Y;

tMaxY += tDeltaY;

Y += stepY;

}

Vector3 nowPos = startPos + new Vector3(stepX \* tMaxX, 0f, stepY \* tMaxY);

int cell = GetCellIndex(nowPos);

if(!checkCanGo(cell, extNum, null))

{

return false;

}

if((nowPos - startPos).sqrMagnitude >= maxDis)

{

return true;

}

}

Debug.Log ("can not find forward pos to Go");

return false;

}

2.如果目标点是不可寻路点，怎么找到最近的一点可行走区域。采用螺旋型搜索，可以最快找到目标。

public Vector3 GetNearestCanGoPos(Vector3 position, int extNum, Dictionary<int, bool> otherAgent)

{

// Save this value off, in case we need to use it to search for a valid location further along in this function.

Vector3 originalPosition = position;

int cellIndex = GetCellIndex(position);

if(cellIndex == SimpleAI.Planning.Node.kInvalidIndex)

{

Debug.Log("targetPos is out of bounds");

return Vector3.zero;

}

int maxCount = 25;

int nowCount = 0;

int len = 1;

if (checkCanGo(cellIndex, extNum, otherAgent))

{

return position;

}

eNeighborDirection dir = eNeighborDirection.kTop;

int index = cellIndex;

while(nowCount < maxCount)

{

nowCount ++;

if(len == 0)

{

len = Mathf.FloorToInt((nowCount + 1) / 2);

switch(dir)

{

case eNeighborDirection.kTop:

dir = eNeighborDirection.kLeft;

break;

case eNeighborDirection.kLeft:

dir = eNeighborDirection.kBottom;

break;

case eNeighborDirection.kBottom:

dir = eNeighborDirection.kRight;

break;

case eNeighborDirection.kRight:

dir = eNeighborDirection.kTop;

break;

}

}

index = GetNeighbor(index, dir);

if(checkCanGo(index, extNum, otherAgent))

{

return GetCellPosition(index);

}

len--;

}

return Vector3.zero;

}

3.一个物体，怎么判断它踩住了多少个格子。特别是不规则的物体。

首先，你要计算出这个物体的边界，剩下就是对边界内的所有格子进行判断，那么我们怎么知道这个格子到底是被物体踩住了，还是空心的呢?

利用射线，可以很快得出结果：

public int[] GetObstructedCells(Grid grid, out int numObstructedCells)

{

numObstructedCells = 0;

if ( collider == null )

{

return null;

}

Bounds bounds = collider.bounds;

bounds.Expand(m\_scale);

// lowerLeftPos Represents the center of the first cell that is covered, in the lower left corner. We march right and up

// from this cell.

Vector3 upperLeftPos = new Vector3( bounds.min.x, grid.Origin.y, bounds.max.z );

Vector3 upperRightPos = new Vector3( bounds.max.x, grid.Origin.y, bounds.max.z );

Vector3 lowerLeftPos = new Vector3( bounds.min.x, grid.Origin.y, bounds.min.z );

Vector3 lowerRightPos = new Vector3( bounds.max.x, grid.Origin.y, bounds.min.z );

Vector3 horizDir = (upperRightPos - upperLeftPos).normalized;

Vector3 vertDir = (upperLeftPos - lowerLeftPos).normalized;

float horizLength = bounds.size.x;

float vertLength = bounds.size.z;

UpdateObstructedCellPool(grid);

// Determine which cells are actually obstructed

for ( int rowCount = 0; rowCount < m\_numObstructedCellPoolRows; rowCount++ )

{

float currentVertLength = rowCount \* grid.CellSize;

for ( int colCount = 0; colCount < m\_numObstructedCellPoolColumns; colCount++ )

{

float currentHorizLength = colCount \* grid.CellSize;

Vector3 testPos = lowerLeftPos + horizDir \* currentHorizLength + vertDir \* currentVertLength;

testPos.x = Mathf.Clamp(testPos.x, bounds.min.x, bounds.max.x);

testPos.z = Mathf.Clamp(testPos.z, bounds.min.z, bounds.max.z);

if ( grid.IsInBounds(testPos) )

{

Vector3 above = testPos + (Vector3.up \* 0.5f);

Vector3 below = testPos + (Vector3.down \* 0.5f);

if (Physics.CheckCapsule(above, below, grid.CellSize / 2f - 0.1f, 1 << LayerMask.NameToLayer("Default")))

{

int obstructedCellIndex = grid.GetCellIndex(testPos);

m\_obstructedCellPool[numObstructedCells] = obstructedCellIndex;

numObstructedCells++;

}

}

if ( currentHorizLength > horizLength )

{

break;

}

}

if ( currentVertLength > vertLength )

{

break;

}

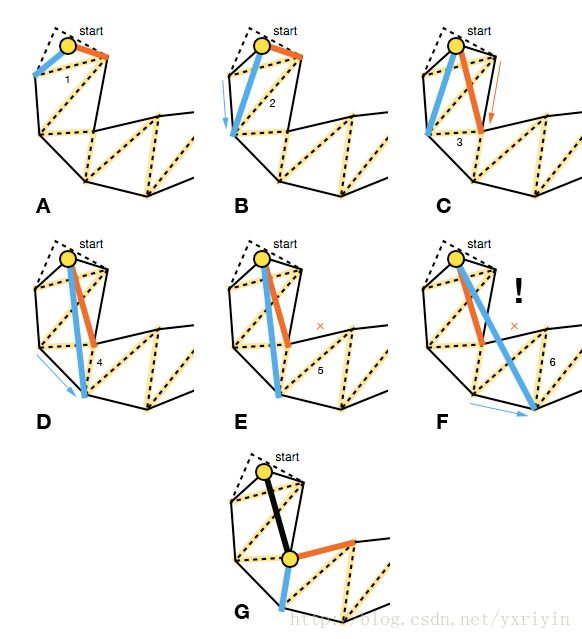
}

return m\_obstructedCellPool;

}

漏斗平滑算法

原理很简单，就是判断下一个点是在漏斗范围内，就直接移动过去。如果出了范围，那么就设置成一个点，重新一轮漏斗。



用Unity3D 如果你没有碰到摄像机抖动的问题，那么你是幸运的。我相信如果你的摄像机控制比较复杂，或者人物移动比较复杂，或多或少总会有这样的疑惑：摄像机好像再抖，或许是模型在抖。总之就是各种抖。当你以为发现问题并解决了，一切都很好，突然有一天，它又开始抖了，而且不止一次这样。我就碰到过这种情况，特此做下总结。

首先你要理解摄像机什么情况下会给你抖动的感觉，简单来说你如果想让摄像机抖动，那么肯定是让摄像机不停变换位置，而且是无规则的。说白了，就是不规则的速度变化会引起抖动。注意这里是相对速度，因为如果绝对速度在变化，但相对速度是一致的，其实你看到的物体是不抖的。这一点在后面的一种情况会用到。接下来就分析下各种情况和原因以级解决方案。

1.摄像机控制很简单，却有大幅度的抖动。

这种情况其实一般不是摄像机控制的问题，而是你摄像机所看物体的策略的问题。首先你要确保摄像机是没帧都更新位置的，你可以放在Update()函数中或者LateUpdate()函数中，切忌不要放在FixedUpdate中去更新位置，因为FixedUpdate不保证没帧都执行，这样就会有断续的现象。然后你要确保摄像机看向的物体的移动是正常的，就是说物体不是在那里抖动的，但摄像机却没有一起抖，这可能和你的逻辑有关。不过这种情况并不常见。大部分时候，相信简单的摄像机控制是最能保证稳定的画面的。

2.摄像机控制简单，大部分时候不抖动，偶尔抖一下，时断时续。

这种情况非常令人困惑，因为理论上应该是不会抖动的，甚至很多时候你要怀疑是不是unity3d自己的问题或者我的帧数不稳定引起的？这里我首先要非常明确一点：Unity3d是一个成熟的商业产品，如果你能够确保正确使用里面的组件和写下正确的代码，那么是不会有这些问题的，包括摄像机抖动和其他一些常见的问题。你首先要看自己的代码和用法是否有问题。

这种情况的原因可能会比较多，我这里只列举我自己碰到的。第一种是因为不必要的平滑算法引起的。你可能会觉得奇怪，平滑算法难道不是为了摄像机不抖而存在的吗？怎么反而会引起摄像机抖动。甚至有的时候你感觉摄像机抖，然后加了平滑算法就不抖了。这里我要解释一下，也是一开始说的，摄像机抖动是基于相对速度的变化不稳定。那么假设你的摄像机本身控制比较简单，其实我们可以基本肯定相对变化速度是一致的。比如你摄像机就是看着一个人，这个去哪摄像机看哪，那么是不是说两人相对速度其实为0。然后你说人抖动怎么办？那么摄像机会一起抖，只不过由于他们相对速度为0，所以你看人是不抖的，但是如果你有场景作为背景，那么你会感觉地面有点抖。这是必须权衡的，因为在场景和人物不同步的情况下，肯定会有一个相对稳定而另一个相对抖动，一般我们和人物一致就好了，毕竟人的视觉肯定是聚焦人物的。那么在这样的基础上，我加了平滑算法，会怎么样呢？平滑无非两种，一种是延迟，一种是根据以前的速度进行加速度计算，然速度变化变慢。不论哪种，在摄像机控制简单的情况下都是画蛇添足。因为首先你已经确保了人和摄像机之间的稳定性。你却加了这种算法，导致了他们不再稳定，而且同时你也无法弥补场景那边被牺牲的抖动，可能反而更抖了。那么照这么说，平滑算法岂不是没用？不是的，后面的情况你会看到平滑算法的引用。只是在摄像机规则很简单的情况下，并不需要额外的平滑算法。

这里还有个小细节，就是你的摄像机一定要先设置好位置，再LookAt,如果你先LookAt,再设置摄像机的位置的话，那么你的摄像机其实LookAt了另一个地方了。这也是引起抖动的原因之一。

3.摄像机控制比较复杂，有时经常抖动，有时候又不抖，神经病一样的。

这种情况就开始复杂起来了。首先我建议你将摄像机改成简单的规则，看是否有抖动。有的话参考1和2里面的先处理。

如果你发现规则简单的时候是不抖动的，但规则复杂的时候抖动，那么你就要好好看看你的规则的代码了。如果你的摄像机移动有速度的概念，一定要将速度和Time.dealtTime联系在一起。也就是这一帧跑了多久时间，这是很重要的，因为每一帧的时间不同，你如果采用相同的位移，其实会引起不同的速度感。

其次，你要明确你的移动算法是基于匀速的，不论是移动还是旋转，都应该是匀速的（也许也可以基于某种速度变化规则，但务必是有规则的运动）。这里特别要注意匀速运动如果到了终点之前，其实是要一个减速的，不然就会过头，过头的话下一帧可能需要回到正确的位置，就会回退了，如此反复，也会抖动。

第三，不建议用ITween这种插件控制摄像机运动，因为它内部的代码其实并没有考虑这些，所以你用ITween导致摄像机抖动是很正常的。当然你图方便然后觉得抖就抖吧那倒是无所谓。

最后，如果还是找不到原因，就排除法，一句句注释掉代码，看到底是哪一句引起了抖动，然后再仔细分析原因。

4. 从正常的逻辑角度来讲摄像机确实会抖动，客观原因。

这种情况也不多，但确实存在。比如你的摄像机的规则是对准多个目标的中心，或者还和力啊，人的转向啊，人的速度变化很突然啊等等有关。如果本身的设计就是导致摄像机抖动的原因，那么这个时候就是平滑算法的用途了，平滑算法本身就是为了让本来变化的不稳定的速度趋于稳定。具体平滑算法有时间再写吧。今天要睡觉了。