3d中的简单数学

一、四元数应用——旋转

1、欧拉角旋转

```
    // The transform to point at the target
public Transform turret;
    public float idleRotationSpeed = 39;
    // If m_Turret rotates freely or only on y;
    public bool onlyYTurretRotation;
    protected virtual void AimTurret(){
                    if (turret == null)
                    {
                            return;
                    }
                    if (m_CurrrentTargetable == null) // do idle rotation
                    {
                            if (m_WaitTimer > 0)
                            {
                                    m_WaitTimer -= Time.deltaTime;
                                    if (m_WaitTimer <= 0)
                                    {
                                            m_CurrentRotationSpeed = (Random.value * 2 - 1)
                                    }
                            }
                            else
                            {
                                    Vector3 euler = turret.rotation.eulerAngles;
                                    euler.x = Mathf.Lerp(Wrap180(euler.x), 0, m_XRotationCor
                                    m_XRotationCorrectionTime = Mathf.Clamp01((m_XRotationCo
                                    euler.y += m_CurrentRotationSpeed * Time.deltaTime;
                                    turret.eulerAngles = euler;
                            }
                    }
                    else
                    {
                            m_WaitTimer = idleWaitTime;
                            Vector3 targetPosition = m_CurrrentTargetable.position;
                            if (onlyYTurretRotation)
                            {
                                    targetPosition.y = turret.position.y;
                            }
                            Vector3 direction = targetPosition - turret.position;
                            Quaternion look = Quaternion.LookRotation(direction, Vector3.up)
                            Vector3 lookEuler = look.eulerAngles;
                            // We need to convert the rotation to a -180/180 wrap so that we
                            float x = Wrap180(lookEuler.x);
                            lookEuler.x = Mathf.Clamp(x, turretXRotationRange.x, turretXRota
                            look.eulerAngles = lookEuler;
                            turret.rotation = look;
                    }
    }
    static float Wrap180(float angle)
            {
                    angle %= 360;
                    if (angle < -180)
```

```
                {
                        angle += 360;
                }
                else if (angle > 180)
                {
                        angle -= 360;
                }
                return angle;
        }
```

2、绕某个轴（自定义）旋转

```
    public override void Launch(Targetable enemy, GameObject attack, Transform firingPoint)
        {
                var homingMissile = attack.GetComponent<HomingLinearProjectile>();
                if (homingMissile == null)
                {
                        Debug.LogError("No HomingLinearProjectile attached to attack obj
                        return;
                }
                Vector3 startingPoint = firingPoint.position;
                Vector3 targetPoint = Ballistics.CalculateLinearLeadingTargetPoint(
                        startingPoint, enemy.position,
                        enemy.velocity, homingMissile.startSpeed,
                        homingMissile.acceleration);
                homingMissile.SetHomingTarget(enemy);
                var attackAffector = GetComponent<AttackAffector>();
                Vector3 direction = attackAffector.towerTargetter.turret.forward;
                Vector3 binormal = Vector3.Cross(direction, Vector3.up);
                Quaternion rotation = Quaternion.AngleAxis(fireVectorXRotationAdjustment
                Vector3 adjustedFireVector = rotation * direction;
                homingMissile.FireInDirection(startingPoint, adjustedFireVector);
                PlayParticles(fireParticleSystem, startingPoint, targetPoint);
        }
```

3、绕鼠标点（屏幕坐标系中的某个点）旋转

```
    void Turning ()
    {
        #if !MOBILE_INPUT
        // Create a ray from the mouse cursor on screen in the direction of the camera.
        Ray camRay = Camera.main.ScreenPointToRay (Input.mousePosition);
        // Create a RaycastHit variable to store information about what was hit by the ray.
        RaycastHit floorHit;
        // Perform the raycast and if it hits something on the floor layer...
        if(Physics.Raycast (camRay, out floorHit, camRayLength, floorMask))
        {
            // Create a vector from the player to the point on the floor the raycast from th
            Vector3 playerToMouse = floorHit.point - transform.position;
            // Ensure the vector is entirely along the floor plane.
            playerToMouse.y = 0f;
            // Create a quaternion (rotation) based on looking down the vector from the play
            Quaternion newRotatation = Quaternion.LookRotation (playerToMouse);
            // Set the player's rotation to this new rotation.
            playerRigidbody.MoveRotation (newRotatation);
        }
}
```

4、绕X、Y、Z一轴旋转

```
    private void Turn ()
    {
        // Determine the number of degrees to be turned based on the input, speed and time b
        float turn = m_TurnInputValue * m_TurnSpeed * Time.deltaTime;
        // Make this into a rotation in the y axis.
        Quaternion turnRotation = Quaternion.Euler (0f, turn, 0f);
        // Apply this rotation to the rigidbody's rotation.
        m_Rigidbody.MoveRotation (m_Rigidbody.rotation * turnRotation);
    }
```

二、摄像机应用

1、移动和缩放

```
private void Move ()
{
    // Find the average position of the targets.
    FindAveragePosition ();
    // Smoothly transition to that position.
    transform.position = Vector3.SmoothDamp(transform.position, m_DesiredPosition, ref m
}
private void FindAveragePosition ()
{
    Vector3 averagePos = new Vector3 ();
    int numTargets = 0;
    // Go through all the targets and add their positions together.
    for (int i = 0; i < m_Targets.Length; i++)
    {
        // If the target isn't active, go on to the next one.
        if (!m_Targets[i].gameObject.activeSelf)
            continue;
        // Add to the average and increment the number of targets in the average.
        averagePos += m_Targets[i].position;
        numTargets++;
    }
    // If there are targets divide the sum of the positions by the number of them to fir
    if (numTargets > 0)
        averagePos /= numTargets;
    // Keep the same y value.
    averagePos.y = transform.position.y;
    // The desired position is the average position;
    m_DesiredPosition = averagePos;
}
private void Zoom ()
{
    // Find the required size based on the desired position and smoothly transition to t
    float requiredSize = FindRequiredSize();
    m_Camera.orthographicSize = Mathf.SmoothDamp (m_Camera.orthographicSize, requiredSiz
}
private float FindRequiredSize ()
{
    // Find the position the camera rig is moving towards in its local space.
    Vector3 desiredLocalPos = transform.InverseTransformPoint(m_DesiredPosition);
    // Start the camera's size calculation at zero.
    float size = 0f;
    // Go through all the targets...
    for (int i = 0; i < m_Targets.Length; i++)
    {
        // ... and if they aren't active continue on to the next target.
        if (!m_Targets[i].gameObject.activeSelf)
            continue;
        // Otherwise, find the position of the target in the camera's local space.
        Vector3 targetLocalPos = transform.InverseTransformPoint(m_Targets[i].position);
        // Find the position of the target from the desired position of the camera's loc
        Vector3 desiredPosToTarget = targetLocalPos - desiredLocalPos;
```

```
            // Choose the largest out of the current size and the distance of the tank 'up'
            size = Mathf.Max(size, Mathf.Abs(desiredPosToTarget.y));
            // Choose the largest out of the current size and the calculated size based on t
            size = Mathf.Max(size, Mathf.Abs(desiredPosToTarget.x) / m_Camera.aspect);
        }
        // Add the edge buffer to the size.
        size += m_ScreenEdgeBuffer;
        // Make sure the camera's size isn't below the minimum.
        size = Mathf.Max (size, m_MinSize);
        return size;
    }
    private void SetCameraTargets()
    {
        // Create a collection of transforms the same size as the number of tanks.
        Transform[] targets = new Transform[m_Tanks.Length];
        // For each of these transforms...
        for (int i = 0; i < targets.Length; i++)
        {
            // ... set it to the appropriate tank transform.
            targets[i] = m_Tanks[i].m_Instance.transform;
        }
        // These are the targets the camera should follow.
        m_CameraControl.m_Targets = targets;
    }
```

2、跟随

```
        public Transform target;            // The position that that camera will be fol
    public float smoothing = 5f;        // The speed with which the camera will be following
    Vector3 offset;                     // The initial offset from the target.
    void Start ()
    {
        // Calculate the initial offset.
        offset = transform.position - target.position;
    }
    void FixedUpdate ()
    {
        // Create a postion the camera is aiming for based on the offset from the target.
        Vector3 targetCamPos = target.position + offset;
        // Smoothly interpolate between the camera's current position and it's target positi
        transform.position = Vector3.Lerp (transform.position, targetCamPos, smoothing * Tim
    }
```