PIMP UP YOUR BLENDS

through custom biomechanical Anim Nodes

ZONE OF TRUISM
#warmup

BLENDING SKELETAL ANIMATION

The seamless merging of multiple animations by interpolating bone movements to create smooth, natural transitions.

ZONE OF TRUISM

#warmup

Criteria for Animation Blending:

- Multiple Animations Combination.
- Transformations Interpolation.
- Smooth Transitions.
- Blend Factor (Weighting).
- Contextual Adaptation.
- Influence on Multiple Bones or Body Parts.
- Time-based or Parameter-driven.

11 bit studios

ZONE OF TRUISM

#warmup

BLENDS TYPES

ZONE OF TRUISM

#warmup

STANDARD BLENDS

INERTIALIZATION

ZONE OF TRUISM

#warmup

Bilinear Interpolation

Linear Interpolation

Ease In/Out

STANDARD BLENDS

Additive Blending

Aim Offset Blending

Layered Blend Per Bone

ZONE OF TRUISM

#warmup

STANDARD BLENDS

$$\text{Lerp}(A, B, \alpha) = (1 - \alpha) \cdot A + \alpha \cdot B$$

$$\text{Bilinear}(A, B, C, D, u, v) = (1 - u)(1 - v) \cdot A + u(1 - v) \cdot B + (1 - u)v \cdot C + uv \cdot D$$

$$\text{Additive Blend}(Base, Additive) = Base + \Delta\text{Pose}(Additive)$$

$$\text{Blend}(A_{bone}, B_{bone}, \alpha) = (1 - \alpha) \cdot A_{bone} + \alpha \cdot B_{bone}$$

$$\text{EaseInOut}(\alpha) = 3\alpha^2 - 2\alpha^3$$

$$\text{Aim Blend} = \sum_{i=1}^{n} (w_i \cdot A_i)$$

11 bit studios

ZONE OF TRUISM
#warmup

STANDARD BLENDS

Blend Space 1D

Blend Space 2D

Layered Blend Per Bone

Blend Poses by Bool

Blend Poses by Int

Blend Poses by Enum

Blend Nodes

Transition Rules

AnimMontage Blending

Time-Based Blending

Pose Asset Blend

Additive Animations

Aim Offset

ZONE OF TRUISM

#warmup

INERTIALIZATION

ZONE OF TRUISM

#warmup

Velocity smoothing

$$P_{blend}(t) = P_{start} + V_{start} \cdot t + \frac{1}{2} \cdot a \cdot t^2$$

Where:

$P_{blend}(t)$ is the position of the bone at time t during the blend.

P_{start} is the starting position of the bone at the time the blend begins.

V_{start} is the initial velocity of the bone at the start of the blend.

a is the acceleration applied to smooth the movement.

t is the time since the blend began.

ZONE OF TRUISM

#warmup

Inertialization

$$V(t) = V_{start} \cdot \left(1 - \frac{t}{T_{blend}}\right)$$

Where:

$V(t)$ is the velocity at time t.

V_{start} is the initial velocity of the bone at the start of the blend.

T_{blend} is the total blend time for the transition.

BIOMECHANICS MODELS

- Hill-Type Muscle.
- Huxley Muscle Contraction.
- Zajac Muscle-Tendon.
- Delp Model.
- Hatze Model.
- Chung Dynamic Muscle.
- Gait Model (Winter’s Model).
- Segmental Inverse Dynamics.

BIOMECHANICS MODELS

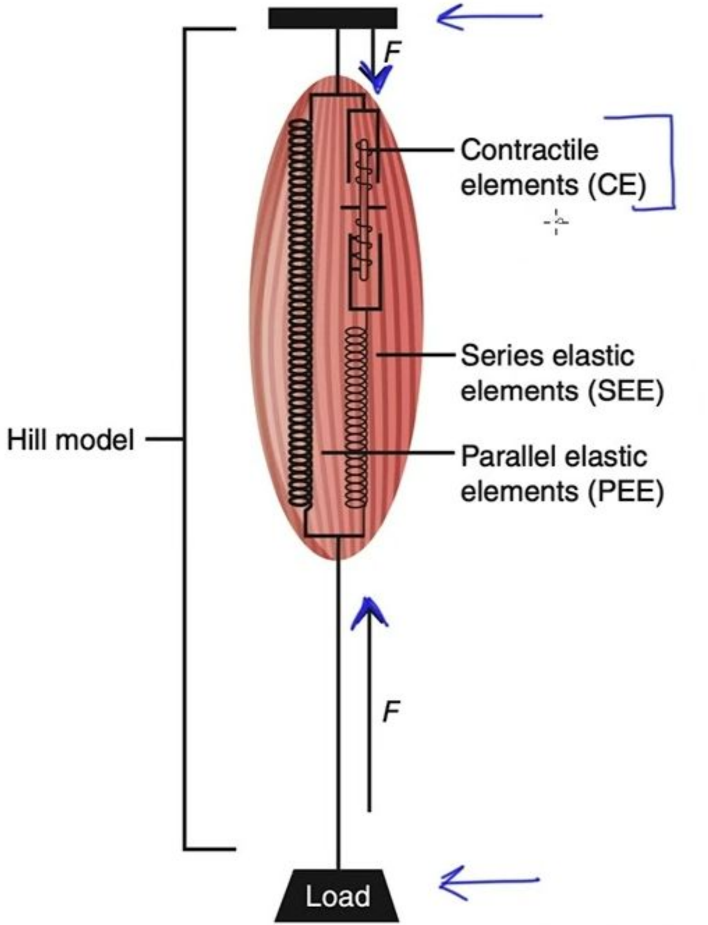
#winner

Hill-Type Muscle Model

A simplified biomechanical model that describes muscle behavior using relationships between muscle force, length, and velocity.

It consists of a contractile element (muscle fibers) and elastic components, making it widely used for simulating muscle dynamics.

11 bit studios



F
Contractile elements (CE)
Series elastic elements (SEE)
Hill model
Parallel elastic elements (PEE)
F
Load

Hill-Type Muscle Model

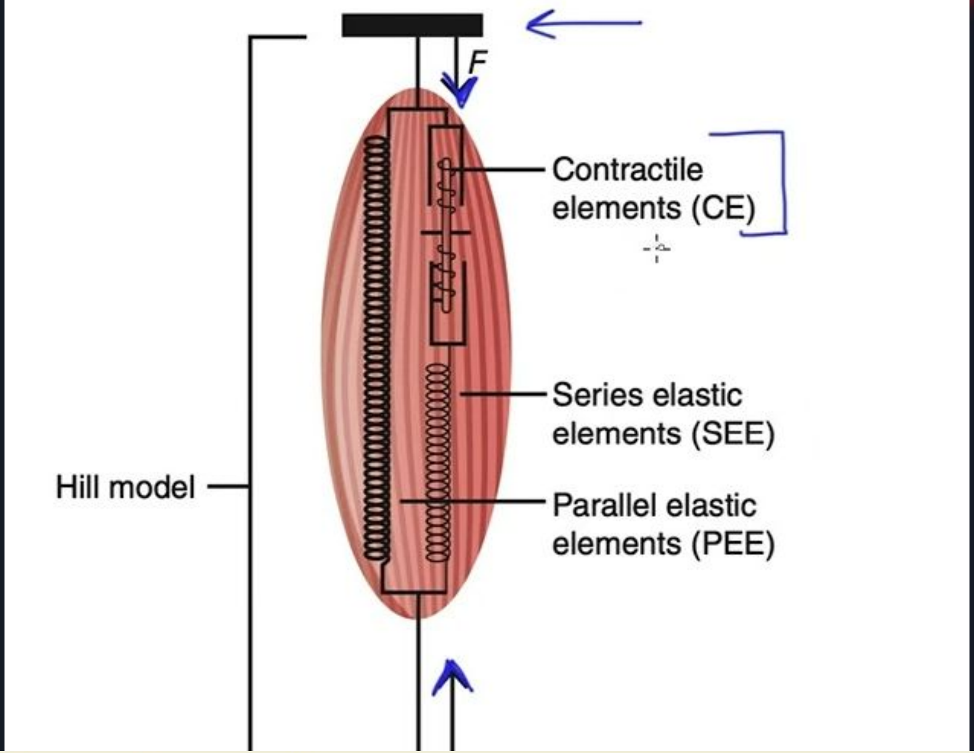
$$F = F_{CE} + F_{PEE} + F_{SEE}$$

F is the total muscle force.

F_{CE} is the force produced by the contractile element of the muscle.

F_{PEE} is the force generated by the parallel elastic element.

F_{SEE} is the force generated by the series elastic element.



$$F = F_{CE} + F_{PEE} + F_{SEE}$$

F is the total muscle force.
F_{CE} is the force produced by the contractile element of the muscle.
F_{PEE} is the force generated by the parallel elastic element.
F_{SEE} is the force generated by the series elastic element.

Hill-Type Muscle Model

$$F = F_{max} \cdot a \cdot \left(1 - \frac{l_0}{l}\right) \cdot e^{-\left(\frac{l_0}{l}\right)^2} + k \cdot (l_0 - l)$$

F is the total muscle force.

F_{max} is the maximum force output of the muscle.

a is the activation level.

l_0 is the optimal length of the muscle.

l is the current length of the muscle.

k is the passive elastic stiffness.

Hill-Type Muscle Model

$$F = F_{CE} + F_{PEE} + F_{SEE}$$

$$F = \underbrace{\boxed{F_{max} \cdot a \cdot \left(1 - \frac{l_0}{l}\right) \cdot e^{-\left(\frac{l_0}{l}\right)^2}}}_{F_{CE}} + \underbrace{\boxed{k \cdot (l_0 - l)}}_{F_{PEE}} + F_{SEE}$$

STEP BY STEP

1. Create dedicated module (eg. plugin).
2. Create UDataAsset for Hill Model Configuration.
3. Define Utility Class for Hill Model Calculations.
4. Get few poses (e.g. for fatigue or pull object).
5. Create for that poses dedicated Hill Model Configs.
6. Implement Animation Node for Requested Blend.
7. Integrate Request Logic Using IGraphMessage.
8. Setup ABP.

Data Asset

UHillModelBasedBlendDataAsset

```
UCLASS(BlueprintType)
class BIOMECHANICALANIMATION_API UBA1HillModelBasedBlendDataAsset : public UPrimaryDataAsset
{
    GENERATED_BODY()

public:
    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "HillModel")
    TArray<FBA1HillBoneParameters> JointParameters;

    const FBA1HillBoneParameters* GetJointParams(const FName& JointName) const
    {
        return JointParameters.FindByPredicate([&](const FBA1HillBoneParameters& Params)
            {
                return Params.JointName == JointName;
            });
    }
};
```

FBAHillBoneParameters

Activation

MuscleLength

ContractionVelocity

MuscleMaxVelocity

PassiveElasticStiffness

TendonStiffness

TendonLength

MomentArmLength

TranslationalSensitivity

UHillModelBasedBlendDataAsset

```
USTRUCT(BlueprintType)
struct FBA1HillBoneParameters
{
    GENERATED_BODY()

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "HillModel")
    FName JointName;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "HillModel")
    float Activation;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "HillModel")
    float MuscleLength;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "HillModel")
    float ContractionVelocity;

    UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "HillModel")
    float MuscleMaxVelocity;
```

UHillModelBasedBlendDataAsset

List of Available Databases

- OpenSim
- PhysioNet
- CMU Graphics Lab Motion Capture Database
- Kinect and Leap Motion Data
- Biomechanics of Human Movement

UHillModelBasedBlendDataAsset

Parameter	Left Foot	Left Knee	Left Hip
Activation	0.8	0.9	1.0
Muscle Length	0.15 m	0.20 m	0.25 m
Contraction Velocity	0.5 m/s	0.6 m/s	0.7 m/s
Muscle Max Velocity	2.0 m/s	2.5 m/s	3.0 m/s
Passive Elastic Stiffness	4000 N/m	5000 N/m	6000 N/m
Tendon Stiffness	3000 N/m	3500 N/m	4000 N/m
Tendon Length	0.10 m	0.12 m	0.15 m
Moment Arm Length	0.05 m	0.06 m	0.07 m
Translational Sensitivity	1.0	1.1	1.2

Biomechanical Calculations

UBiomechanicalUtils

```cpp
UCLASS()
class BIOMECHANICALANIMATION_API UBA1BiomechanicalUtils : public UObject
{
    GENERATED_BODY()

public:
    // High-level function to calculate muscle force
    UFUNCTION(BlueprintCallable, Category = "Biomechanics")
    static float CalculateMuscleForce(const FName& JointName, const UBA1HillModelBasedBlendDataAsset* HillModelConfig);

    // Force-Length relationship calculation
    UFUNCTION(BlueprintCallable, Category = "Biomechanics")
    static float CalculateForceLengthFactor(float MuscleLength);

    // Force-Velocity relationship calculation
    UFUNCTION(BlueprintCallable, Category = "Biomechanics")
    static float CalculateForceVelocityFactor(float ContractionVelocity, float MuscleMaxVelocity);

    // Passive Elastic Element (PEE) calculation
    UFUNCTION(BlueprintCallable, Category = "Biomechanics")
    static float CalculateParallelElasticForce(float MuscleLength, float PassiveElasticStiffness);

    // Series Elastic Element (SEE) calculation
    UFUNCTION(BlueprintCallable, Category = "Biomechanics")
    static float CalculateSeriesElasticForce(float TendonLength, float TendonStiffness);
};
```

UBiomechanicalUtils

$$F = F_{max} \cdot a \cdot \left(1 - \frac{l_0}{l}\right) \cdot e^{-\left(\frac{l_0}{l}\right)^2} + k \cdot (l_0 - l)$$

Where:

F is the total muscle force.

F_{max} is the maximum force output of the muscle.

a is the activation level.

l_0 is the optimal length of the muscle.

l is the current length of the muscle.

k is the passive elastic stiffness.

UBiomechanicalUtils

```
// Main function to calculate total muscle force using Hill's model
float UBA1BiomechanicalUtils::CalculateMuscleForce(const FName& JointName, const UBA1HillModelBasedBlendDataAsset* HillMod
{
    const FBA1HillBoneParameters* JointParams = HillModelConfig->JointParameters.FindByPredicate([&](const FBA1HillBonePa
        {
            return Params.JointName == JointName;
        });

    if (!JointParams)
    {
        return 0.f;
    }

    // Calculate the force components based on Hill's muscle model
    const float ForceLengthFactor = CalculateForceLengthFactor(JointParams->MuscleLength);
    const float ForceVelocityFactor = CalculateForceVelocityFactor(JointParams->ContractionVelocity, JointParams->MuscleMa
    const float ParallelElasticForce = CalculateParallelElasticForce(JointParams->MuscleLength, JointParams->PassiveElast
    const float SeriesElasticForce = CalculateSeriesElasticForce(JointParams->TendonLength, JointParams->TendonStiffness);

    // Return total muscle force
    return JointParams->Activation * ForceLengthFactor * ForceVelocityFactor + ParallelElasticForce + SeriesElasticForce;
}
```

UBiomechanicalUtils

```
// Calculate the force components based on Hill's muscle model
const float ForceLengthFactor = CalculateForceLengthFactor(JointParams->MuscleLength);
const float ForceVelocityFactor = CalculateForceVelocityFactor(JointParams->ContractionVelocity, JointParams->MuscleMaxVelocity);
const float ParallelElasticForce = CalculateParallelElasticForce(JointParams->MuscleLength, JointParams->PassiveElasticStiffness);
const float SeriesElasticForce = CalculateSeriesElasticForce(JointParams->TendonLength, JointParams->TendonStiffness);

// Return total muscle force
return JointParams->Activation * ForceLengthFactor * ForceVelocityFactor + ParallelElasticForce + SeriesElasticForce;
```

$$F = F_{max} \cdot a \cdot \left(1 - \frac{l_0}{l}\right) \cdot e^{-\left(\frac{l_0}{l}\right)^2} + k \cdot (l_0 - l)$$

UBiomechanicalUtils

```
// Calculate the force components based on Hill's muscle model
const float ForceLengthFactor = CalculateForceLengthFactor(JointParams->MuscleLength);
const float ForceVelocityFactor = CalculateForceVelocityFactor(JointParams->ContractionVelocity, JointParams->MuscleMaxVelocity);
const float ParallelElasticForce = CalculateParallelElasticForce(JointParams->MuscleLength, JointParams->PassiveElasticStiffness);
const float SeriesElasticForce = CalculateSeriesElasticForce(JointParams->TendonLength, JointParams->TendonStiffness);

// Return total muscle force
return JointParams->Activation * ForceLengthFactor * ForceVelocityFactor + ParallelElasticForce + SeriesElasticForce;
```

$$F = F_{max} \boxed{\cdot\, a\, \cdot} \left(1 - \frac{l_0}{l}\right) \cdot e^{-\left(\frac{l_0}{l}\right)^2} + k \cdot (l_0 - l)$$

UBiomechanicalUtils

```
// Calculate the force components based on Hill's muscle model
const float ForceLengthFactor = CalculateForceLengthFactor(JointParams->MuscleLength);
const float ForceVelocityFactor = CalculateForceVelocityFactor(JointParams->ContractionVelocity, JointParams->MuscleMaxVelocity);
const float ParallelElasticForce = CalculateParallelElasticForce(JointParams->MuscleLength, JointParams->PassiveElasticStiffness);
const float SeriesElasticForce = CalculateSeriesElasticForce(JointParams->TendonLength, JointParams->TendonStiffness);

// Return total muscle force
return JointParams->Activation * ForceLengthFactor * ForceVelocityFactor + ParallelElasticForce + SeriesElasticForce;
```

$$F = F_{max} \cdot a \cdot \left(1 - \frac{l_0}{l}\right) \cdot e^{-\left(\frac{l_0}{l}\right)^2} + k \cdot (l_0 - l)$$

UBiomechanicalUtils

```
// Calculate the force components based on Hill's muscle model
const float ForceLengthFactor = CalculateForceLengthFactor(JointParams->MuscleLength);
const float ForceVelocityFactor = CalculateForceVelocityFactor(JointParams->ContractionVelocity, JointParams->MuscleMaxVelocity);
const float ParallelElasticForce = CalculateParallelElasticForce(JointParams->MuscleLength, JointParams->PassiveElasticStiffness);
const float SeriesElasticForce = CalculateSeriesElasticForce(JointParams->TendonLength, JointParams->TendonStiffness);

// Return total muscle force
return JointParams->Activation * ForceLengthFactor * ForceVelocityFactor + ParallelElasticForce + SeriesElasticForce;
```

$$F = F_{max} \cdot a \cdot \left(1 - \frac{l_0}{l}\right) \cdot \boxed{e^{-\left(\frac{l_0}{l}\right)^2}} + k \cdot (l_0 - l)$$

UBiomechanicalUtils

```
// Calculate the force components based on Hill's muscle model
const float ForceLengthFactor = CalculateForceLengthFactor(JointParams->MuscleLength);
const float ForceVelocityFactor = CalculateForceVelocityFactor(JointParams->ContractionVelocity, JointParams->MuscleMaxVelocity);
const float ParallelElasticForce = CalculateParallelElasticForce(JointParams->MuscleLength, JointParams->PassiveElasticStiffness);
const float SeriesElasticForce = CalculateSeriesElasticForce(JointParams->TendonLength, JointParams->TendonStiffness);

// Return total muscle force
return JointParams->Activation * ForceLengthFactor * ForceVelocityFactor + ParallelElasticForce + SeriesElasticForce;
```

$$F = F_{max} \cdot a \cdot \left(1 - \frac{l_0}{l}\right) \cdot e^{-\left(\frac{l_0}{l}\right)^2} + k \cdot (l_0 - l)$$

UBiomechanicalUtils

```
// Calculate the force components based on Hill's muscle model
const float ForceLengthFactor = CalculateForceLengthFactor(JointParams->MuscleLength);
const float ForceVelocityFactor = CalculateForceVelocityFactor(JointParams->ContractionVelocity, JointParams->MuscleMaxVelocity);
const float ParallelElasticForce = CalculateParallelElasticForce(JointParams->MuscleLength, JointParams->PassiveElasticStiffness);
const float SeriesElasticForce = CalculateSeriesElasticForce(JointParams->TendonLength, JointParams->TendonStiffness);

// Return total muscle force
return JointParams->Activation * ForceLengthFactor * ForceVelocityFactor + ParallelElasticForce + SeriesElasticForce;
```

$$F = \underbrace{\boxed{F_{max} \cdot a \cdot \left(1 - \frac{l_0}{l}\right) \cdot e^{-\left(\frac{l_0}{l}\right)^2}}}_{F_{CE}} + \underbrace{\boxed{k \cdot (l_0 - l)}}_{F_{PEE}} + F_{SEE}$$

UBiomechanicalUtils

```
// Force-Length relationship calculation
float UBA1BiomechanicalUtils::CalculateForceLengthFactor(float MuscleLength)
{
    // Gaussian curve representing muscle force-length relationship
    return FMath::Exp(-FMath::Pow((MuscleLength - 1.0f), 2.0f) / 0.45f);
}
```

$$F = F_{max} \cdot a \cdot \left(1 - \frac{l_0}{l}\right) \cdot \boxed{e^{-\left(\frac{l_0}{l}\right)^2}} + k \cdot (l_0 - l)$$

UBiomechanicalUtils

```
// Force-Velocity relationship calculation
float UBA1BiomechanicalUtils::CalculateForceVelocityFactor(float ContractionVelocity, float MuscleMaxVelocity)
{
    // Linear relationship for muscle force-velocity
    return 1.0f - (ContractionVelocity / MuscleMaxVelocity);
}
```

$$F = F_{max} \cdot a \cdot \left(1 - \frac{l_0}{l}\right) \cdot e^{-\left(\frac{l_0}{l}\right)^2} + k \cdot (l_0 - l)$$

UBiomechanicalUtils

```
// Passive Elastic Element (PEE) calculation
float UBA1BiomechanicalUtils::CalculateParallelElasticForce(float MuscleLength, float PassiveElasticStiffness)
{
    // Force exerted by the passive elastic component
    return PassiveElasticStiffness * (MuscleLength - 1.0f);
}
```

$$F = F_{max} \cdot a \cdot \left(1 - \frac{l_0}{l}\right) \cdot e^{-\left(\frac{l_0}{l}\right)^2} + \boxed{k \cdot (l_0 - l)}$$

UBiomechanicalUtils

```
// Series Elastic Element (SEE) calculation
float UBA1BiomechanicalUtils::CalculateSeriesElasticForce(float TendonLength, float TendonStiffness)
{
    // Force exerted by the series elastic component
    return TendonStiffness * (TendonLength - 1.0f);
}
```

$$F = F_{CE} + F_{PEE} + F_{SEE}$$

Custom Anim Nodes

Hill Model Based Fatigue Blend

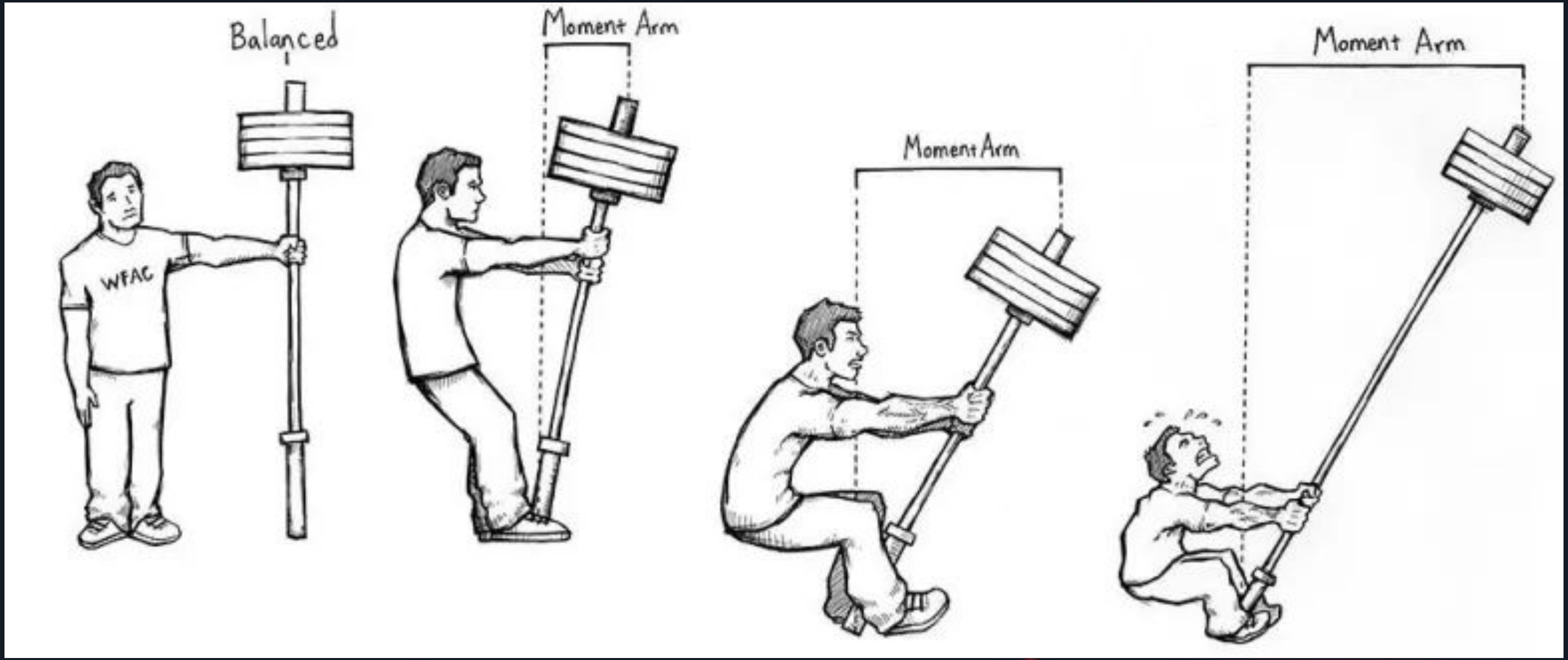
```
// Iterate through specified joint names
for (const FName& JointName : JointNames)
{
    // Get the index of the joint
    int32 SkeletonBoneIndex = Output.AnimInstanceProxy->GetSkelMeshComponent()->GetBoneIndex(JointName);
    FCompactPoseBoneIndex BoneIndex = BoneContainer.GetCompactPoseIndexFromSkeletonIndex(SkeletonBoneIndex);

    // Apply fatigue to the muscle force for this joint
    FTransform NewTransform = Output.Pose[BoneIndex];

    // Calculate the muscle force influenced by fatigue using BiomechanicalUtils
    const FBA1HillBoneParameters& JointParams = *HillModelConfig->GetJointParams(JointName);
    float MuscleForce = UBA1BiomechanicalUtils::CalculateMuscleForce(JointParams);

    // Get the moment arm length from the config for the joint
    float MomentArmLength = HillModelConfig->GetMomentArmLength(JointName); // Zakładając, że masz tę metodę
    // Calculate torque
    float Torque = UBA1BiomechanicalUtils::CalculateTorque(MuscleForce, HillModelConfig->GetMomentArmLength(JointName));
    // Calculate rotation angle based on torque
    float RotationChange = Torque * RotationMultiplier; // Adjust the influence of Torque on rotation
```

Hill Model Based Fatigue Blend



IGraphMessage

IGraphMessage enables communication between animation nodes.

Nodes can send/receive specific messages for updates.

Allows nodes to request actions based on conditions.

Nodes send messages to trigger transitions or blends.

IGraphMessage

Example from UE - Inertialization

```cpp
// Event that can be subscribed to request inertialization-based blends
class IInertializationRequester : public IGraphMessage
{
	DECLARE_ANIMGRAPH_MESSAGE_API(IInertializationRequester, ENGINE_API);

public:
	static ENGINE_API const FName Attribute;

	// Request to activate inertialization for a duration.
	// If multiple requests are made on the same inertialization node, the minimum requested time will be used.
	virtual void RequestInertialization(float InRequestedDuration, const UBlendProfile* InBlendProfile = nullptr) = 0;

	// Request to activate inertialization.
	// If multiple requests are made on the same inertialization node, the minimum requested time will be used.
	ENGINE_API virtual void RequestInertialization(const FInertializationRequest& InInertializationRequest);

	// Add a record of this request
	virtual void AddDebugRecord(const FAnimInstanceProxy& InSourceProxy, int32 InSourceNodeId) = 0;
};
```

IGraphMessage

Example from UE - Inertialization

```
// Inertialization request event bound to a node
class FInertializationRequester : public IInertializationRequester
{
public:
    FInertializationRequester(const FAnimationBaseContext& InContext, FAnimNode_Inertialization* InNode)
        : Node(*InNode)
        , NodeId(InContext.GetCurrentNodeId())
        , Proxy(*InContext.AnimInstanceProxy)
    {}

private:
    // IInertializationRequester interface
    virtual void RequestInertialization(float InRequestedDuration, const UBlendProfile* InBlendProfile) override
    {
        Node.RequestInertialization(InRequestedDuration, InBlendProfile);
    }
```

IGraphMessage

Example from UE - Inertialization

void FAnimNode_Inertialization::Update_AnyThread
Line: 393

```
// Catch the inertialization request message and call the node's RequestInertialization function with the request
UE::Anim::TScopedGraphMessage<UE::Anim::FInertializationRequester> InertializationMessage(Context, Context, this);
```

IGraphMessage

Hill Model Based Requested Blend

```
class IHillModelBlendRequester : public IGraphMessage
{
    DECLARE_ANIMGRAPH_MESSAGE_API(IHillModelBlendRequester, BIOMECHANICALANIMATION_API);

public:
    static BIOMECHANICALANIMATION_API const FName Attribute;

    virtual void RequestHillModelBlend(float InRequestedDuration) = 0;

    virtual void AddDebugRecord(const FAnimInstanceProxy& InSourceProxy, int32 InSourceNodeId) = 0;
};
```

```
void FBA1AnimNode_HillModelFatigue::Update_AnyThread(const FAnimationUpdateContext& Context)
{
   UE::Anim::TScopedGraphMessage<UE::Anim::IHillModelBlendRequester> HillBlendMessage(Context, Context, this);

   // Update the evaluation graph inputs
   GetEvaluateGraphExposedInputs().Execute(Context);
}
```

WHAT NEXT

What Next?

Optimization through parallelization of calculations.

Expansion to other biomechanical models.

Combining this approach with Standard Blends
or Inertialization (an interesting hybrid).

Enhancing configs for animation clusters (e.g., for motion matching).

Adrianna Bielak

Gameplay & Animation
Programmer