thrive
SIGGRAPH2019
LOS ANGELES • 28 JULY – 1 AUGUST

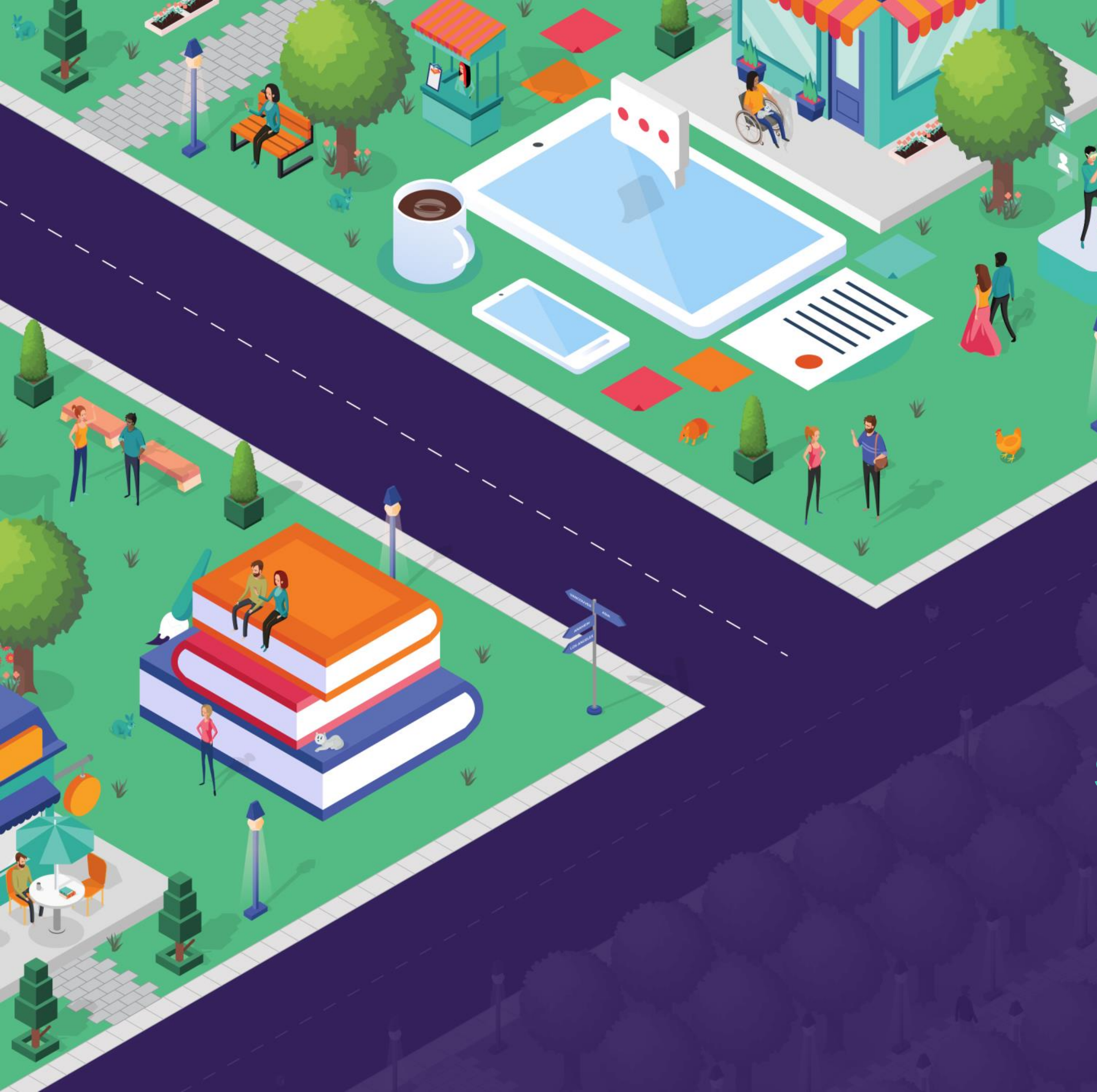
ADVANCED USE CASES FOR ANIMATION RIGGING IN UNITY

SIGGRAPH Studio Workshop

Dave Hunt, Technical Artist, Unity

Olivier Dionne, Animation Developer, Unity

Simon Bouvier-Zappa, Animation Developer, Unity



PART 1

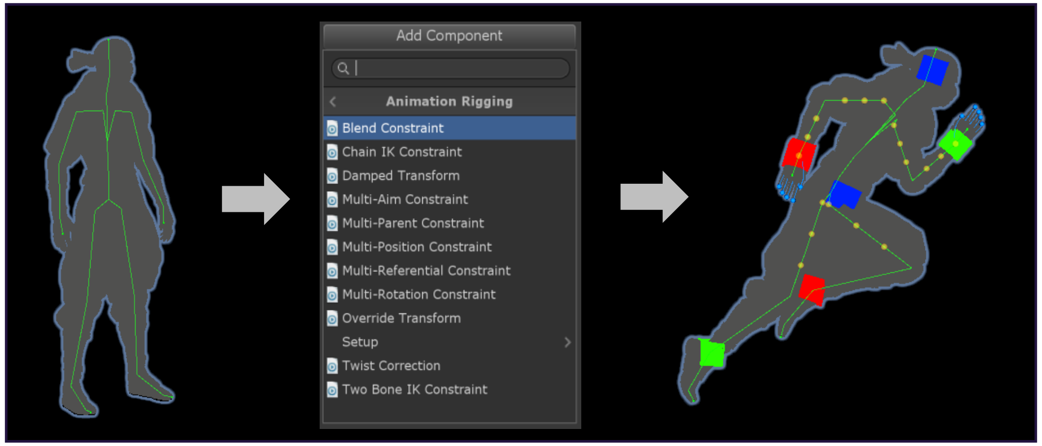
Setting Up Animation Rigs in Unity

Dave Hunt, Technical Artist, Unity

INTRODUCTION TO THE ANIMATION RIGGING PACKAGE

Overview

- The Animation Rigging (Preview) package enables procedural motion for animated skeletons at runtime
- Unity 2019.1 – Runtime Rigging
- Unity 2019.2 – Keyframing in Animation Window and Effectors
  - Enables motion editing workflows in the Unity Editor
- Unity 2019.3 – Keyframing in Timeline
  - Enables multi-track, layered animation authoring

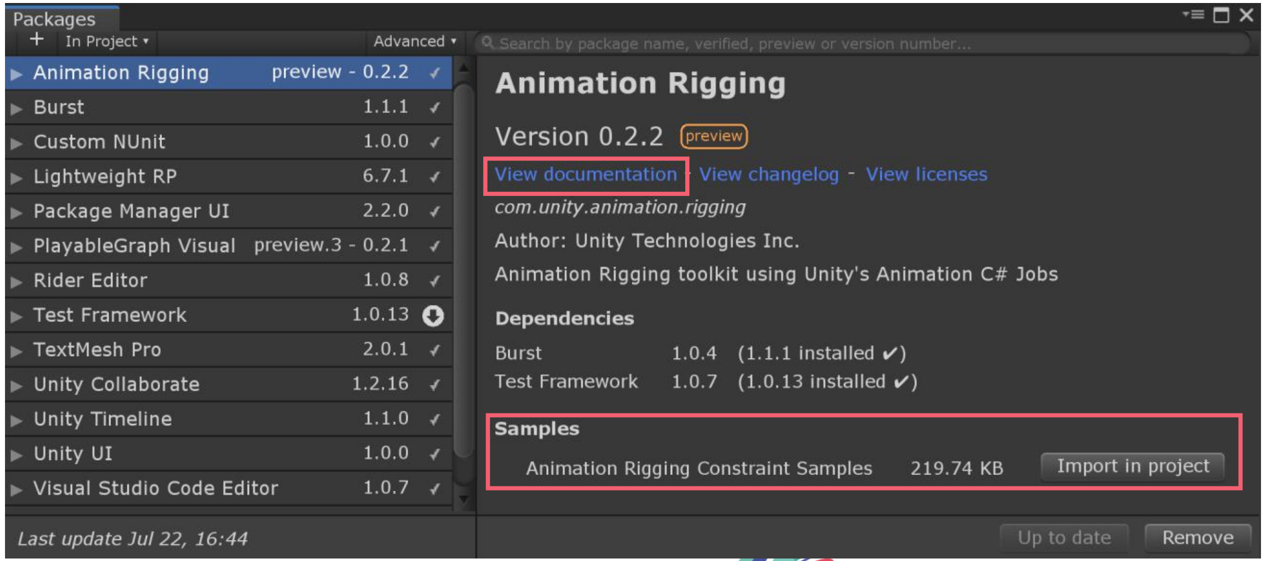


Installation

- One-click installation from Package Manager

Samples and Documentation

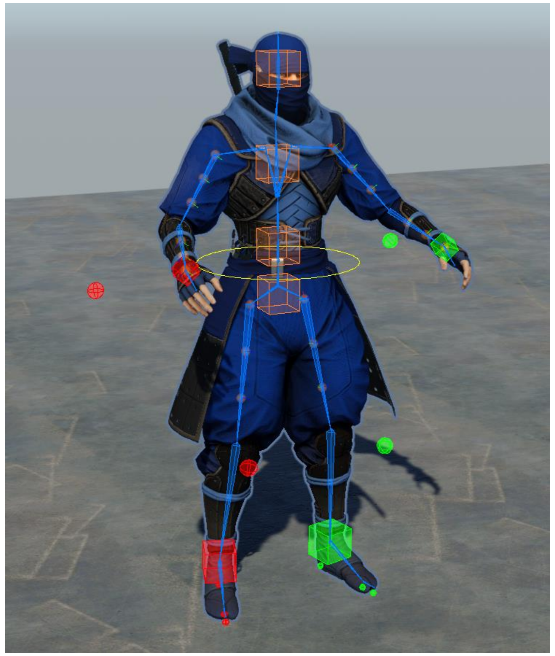
- Links are included in Package Manager



Online docs:
https://docs.unity3d.com/Packages/com.unity.animation.rigging@0.2/manual/index.html

WORKSHOP: SETTING UP A FULL BODY CHARACTER RIG

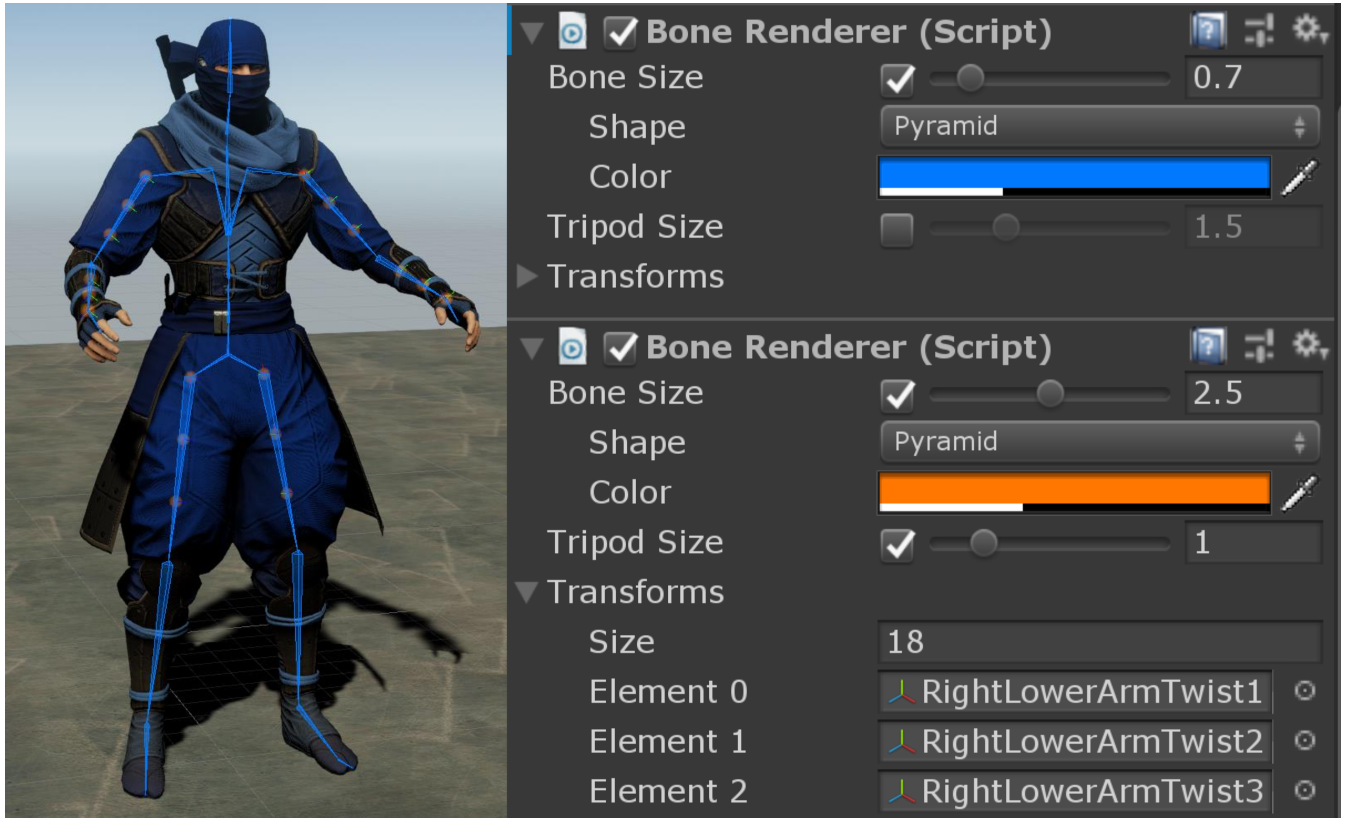
- This rig works for Runtime Rigging and Animation Authoring
  - All characters are playing the same animation clip
  - Animation variations are made using keyframed rig overrides



thrive SIGGRAPH2019 LOS ANGELES • 28 JULY – 1 AUGUST

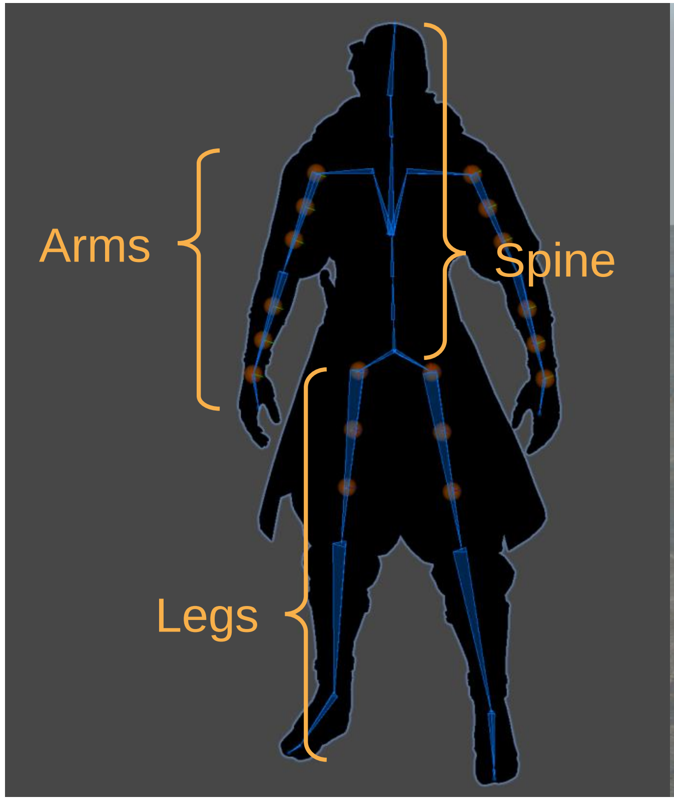
SETTING UP THE SKELETON

- Open the scene: 01_WorkshopStart
- Bone Renderer
  - See and interact with skeletons in the Scene
  - Multiple Bone Renderer components have already been added to the Animator Root
    - One for the main body skeleton
    - Another for the twist bones
  - **Setup instructions** (this is already setup)
    1. Add Bone Renderer component
    2. Add transforms to the list
    3. Customize display options



SETTING UP A CONTROL RIG FOR THE FULL BODY

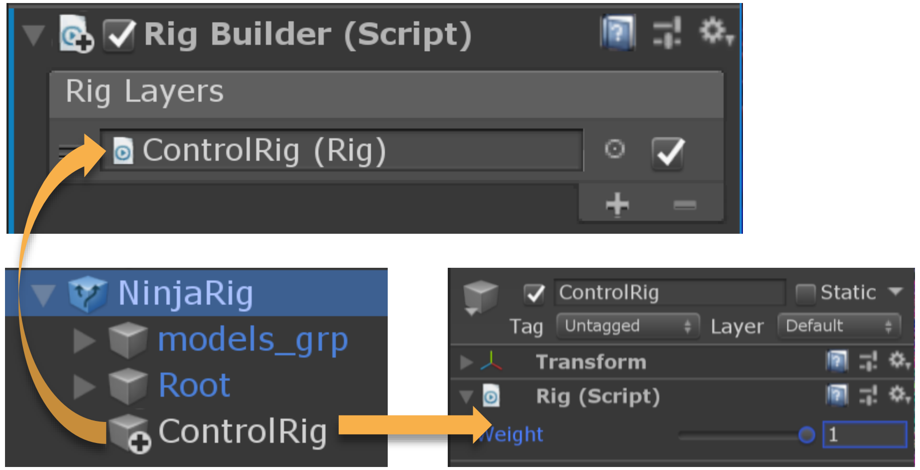
- Organizing the rig in Regions
  - Arms
  - Legs
  - Spine (with head)



SETTING UP A CONTROL RIG FOR THE FULL BODY

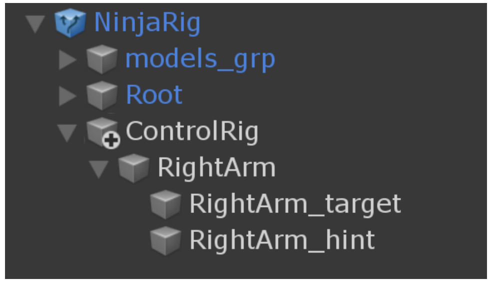
- Create a new Rig for the control rig
  - There can be multiple rigs!

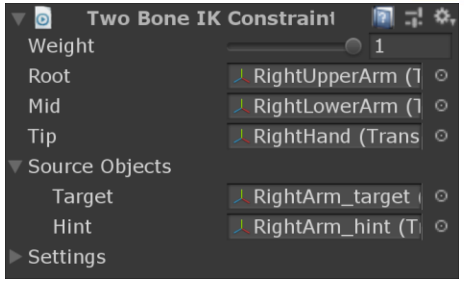
  - **Setup instructions**
    1. Select NinjaRig and add the Rig Builder component
       1. In the Rig Layers list click the + button to add a new entry
    2. Create a new child GameObject and name it ControlRig
       1. Add the Rig component
    3. Drag and drop ControlRig into the Rig Layers list



thrive SIGGRAPH2019 LOS ANGELES • 28 JULY – 1 AUGUST

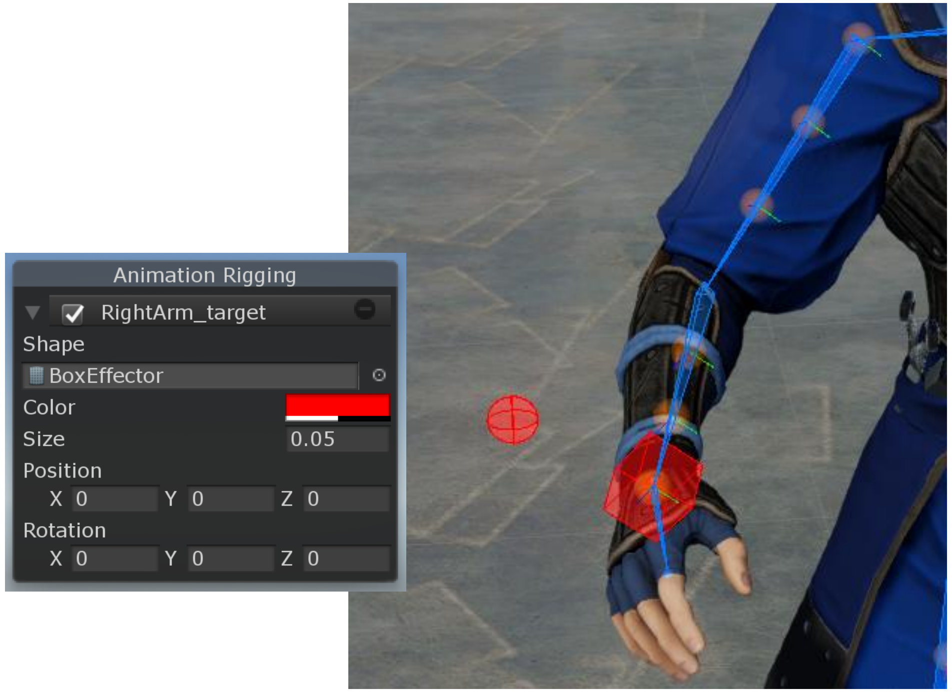
SETTING UP THE ARM REGION – RIGHT SIDE





- Create the Right Arm region
  - **Setup instructions**
    1. Create a new GameObject below ControlRig, name it "RightArm"
    2. Add the TwoBoneIKConstraint component
    3. Add the arm bones for Root, Mid and Tip (see screenshot)
       - Note: it is helpful to lock the Inspector while adding scene objects

    4. Add Effectors
       1. Create two new GameObjects below RightArm and name them "RightArm_target" and "RightArm_hint"
       2. Assign these to the TwoBoneIK – Target and Hint fields
       3. Customize the effector display options
       4. Align the effectors to the skeleton
          1. Select RightArm_target, then ctrl-select the RightHand bone
          2. On the Animation Rigging menu click Align Transform
          3. Also align the RightArm_hint to the RightLowerArm bone



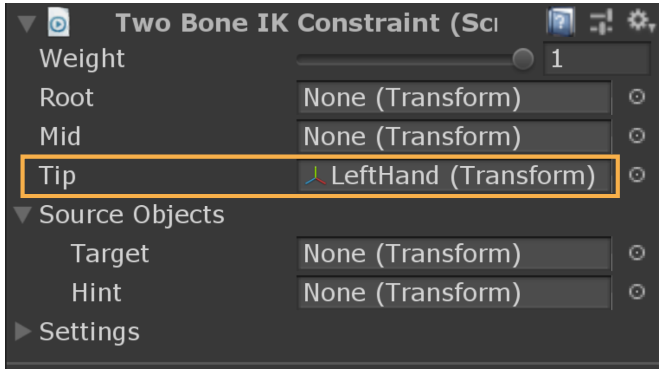
    5. Press Play to see and interact with the Arm IK rig
    6. Or press Preview in Animation Window

thrive SIGGRAPH2019 LOS ANGELES • 28 JULY – 1 AUGUST

SETTING UP THE ARM REGION – LEFT SIDE

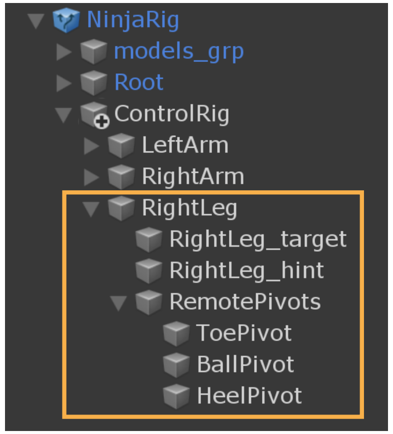
- Create the Left Arm region (auto-setup)
  - Automation script example:
  - Assets/Editor/TwoBoneIKAutoSetup.cs

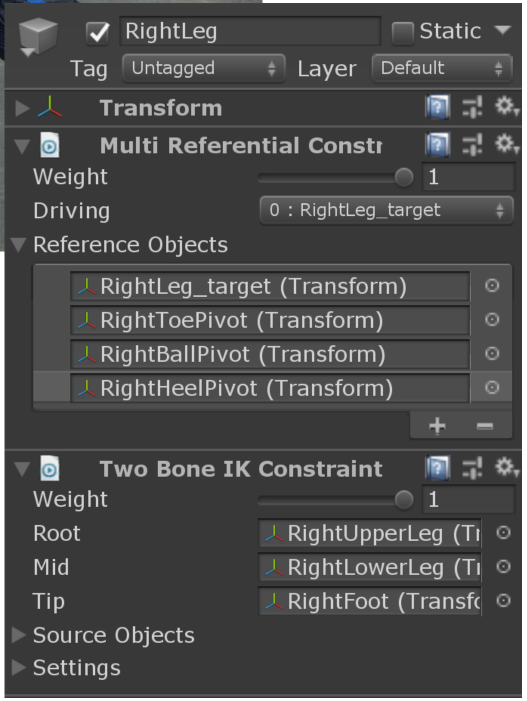
  - **Setup instructions**
    1. Create a new GameObject below ControlRig, name it "LeftArm"
    2. Add the TwoBoneIKConstraint component
    3. Add only the Tip bone
    4. With the TwoBoneIK object selected, run the auto-setup command…
        1. Animation Rigging > Utilities > Auto-Setup TwoBoneIK from Tip Transform
    5. Customize the Effector display options



SETTING UP THE LEG REGION – RIGHT SIDE

- Create the Right Leg region
  - TwoBoneIK with remote pivots using Multi-Referential Constraints
  - **Setup instructions**
    1. Create a GameObject for the leg IK (child of ControlRig, named “RightLeg”)
       1. Add a TwoBoneIK component
       2. Assign the RightFoot bone as Tip and run Auto-Setup TwoBoneIK
       - Animation Rigging > Utilities > Auto-Setup TwoBoneIK from Tip Transform
    2. Add remote pivots for the foot
       1. Create a group GameObject (child of RightLeg named “RemotePivots”)
       2. Create child GameObjects for: ToePivot, BallPivot and HeelPivot
       3. Customize the Effector display options and align them to the character model
    3. Add a Multi-Referential Constraint to the RightLeg
       1. Assign the RightLeg_target and remote pivots as Reference Objects
       2. ***Reorder the components so that Multi-Referential comes before TwoBoneIK
    4. Press Play and interact with the Multi-Referential Constraint
       1. Change the Driving object to switch pivots

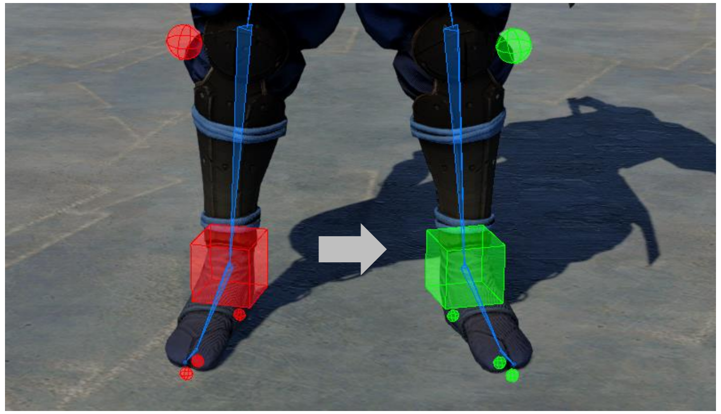


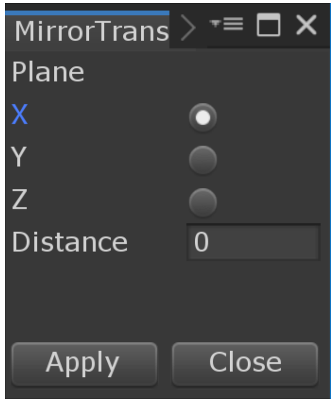


SETTING UP THE LEG REGION – LEFT SIDE

- Copy the Right Leg to the Left side using Prefabs
  - The Prefab system makes it easy to save copies of rig regions. These can be shared with the same skeleton or different ones.

  - **Setup instructions**
    1. Save a prefab of the right leg
       1. Drag and drop the RightLeg into a folder in your Project
    2. Add a copy of the prefab for the left leg
       1. Drag the RightLeg prefab onto ControlRig and rename it to LeftLeg
       2. Assign the TwoBoneIK Tip transform and run Auto-Setup TwoBoneIK
    3. Mirror the remote pivot Effector positions
       1. Animation Rigging menu > Utilities > Mirror Transforms
       2. Customize the Effector display options





thrive
SIGGRAPH2019
LOS ANGELES • 28 JULY – 1 AUGUST

SETTING UP THE SPINE REGION

- Center of Gravity
  - Top-level control for the whole spine
- Pelvis
  - Independent rotation without affecting the upper body
- Torso
  - Top and bottom rotation controls
  - Automatic blended rotation of the middle bone(s)
- Head
  - Single rotation control for the head
  - Automatic blended rotation of the neck

SETTING UP THE SPINE REGION – TORSO AND HEAD

- Want to have:
  - Torso
    - Top and bottom rotation controls
    - Automatically blend rotation of the middle bones
  - Head
    - Head rotation control with automatic blended neck rotation

- Let's develop a new constraint in C#

PART 2

Extending the Animation Rigging package with C#

Olivier Dionne, Animation Developer, Unity

Simon Bouvier-Zappa, Animation Developer, Unity

OPEN RIGGING TOOLKIT

- Each production has different requirements

- Custom constraints can be tailor-made to your specific use cases
  - Potential performance gains
  - Unique interactions or behaviors

- Source of all constraints shipping with the package is available and thus editable/extensible

ANIMATION RIGGING PACKAGE IN A NUTSHELL

- Built on top of Animation C# jobs
  - Blog post : https://blogs.unity3d.com/2018/08/27/animation-c-jobs/
  - Enables you to modify an animation pose on the Animator thread prior to writing it back to the GameObjects
  - Frictionless multi-threaded scheduling performed by the Animator

- Abstracts complexity required to create distributed constraints

- Interactive layering of multiple rigs and constraint combinations

- WYSIWYG constraint scheduling via the hierarchy view

SCRIPTING A CUSTOM CONSTRAINT

Boils down to declaring a **RigConstraint**

```
public class RigConstraint<TJob, TData, TBinder> : MonoBehaviour, IRigConstraint
    where TJob    : struct, IWeightedAnimationJob
    where TData   : struct, IAnimationJobData
    where TBinder : AnimationJobBinder<TJob, TData>, new()
{ }
```

SCRIPTING A CUSTOM CONSTRAINT

Boils down to declaring a **RigConstraint**

```
public class RigConstraint<TJob, TData, TBinder> : MonoBehaviour, IRigConstraint
    where TJob     : struct, IWeightedAnimationJob
    where TData    : struct, IAnimationJobData
    where TBinder : AnimationJobBinder<TJob, TData>, new()
{ }
```

SCRIPTING A CUSTOM CONSTRAINT

Boils down to declaring a **RigConstraint**

```
public class RigConstraint<TJob, TData, TBinder> : MonoBehaviour, IRigConstraint
    where TJob     : struct, IWeightedAnimationJob
    where TData    : struct, IAnimationJobData
    where TBinder : AnimationJobBinder<TJob, TData>, new()
{ }
```

SCRIPTING A CUSTOM CONSTRAINT

Boils down to declaring a **RigConstraint**

```
public class RigConstraint<TJob, TData, TBinder> : MonoBehaviour, IRigConstraint
    where TJob     : struct, IWeightedAnimationJob
    where TData    : struct, IAnimationJobData
    where TBinder : AnimationJobBinder<TJob, TData>, new()
{ }
```

SCRIPTING A CUSTOM CONSTRAINT

Boils down to declaring a **RigConstraint**

```
public class HelloWorld : RigConstraint<HelloWorldJob, HelloWorldData, HelloWorldBinder>
{ }
```

SCRIPTING A CUSTOM CONSTRAINT

```
public class HelloWorld : RigConstraint<HelloWorldJob, HelloWorldData, HelloWorldBinder>
{ }
```

HelloWorldJob is an IWeightedAnimationJob

```
public interface IWeightedAnimationJob : IAnimationJob
{
    FloatProperty jobWeight { get; set; }
}
```

An IAnimationJob containing an auto populated weight property

SCRIPTING A CUSTOM CONSTRAINT

```
public class HelloWorld : RigConstraint<HelloWorldJob, HelloWorldData, HelloWorldBinder>
{ }
```

HelloWorldJob is an IWeightedAnimationJob

```
[BurstCompile]
public struct HelloWorldJob : IWeightedAnimationJob
{
    public ReadWriteTransformHandle constrained;
    public ReadOnlyTransformHandle source;

    public FloatProperty jobWeight { get; set; }

    public void ProcessRootMotion(AnimationStream stream) { }

    public void ProcessAnimation(AnimationStream stream)
    {
        float w = jobWeight.Get(stream);
        if (w > 0f)
        {
            constrained.SetPosition(
                stream,
                math.lerp(constrained.GetPosition(stream), -source.GetPosition(stream), w)
                );
        }
    }
}
```

SCRIPTING A CUSTOM CONSTRAINT

```
public class HelloWorld : RigConstraint<HelloWorldJob, HelloWorldData, HelloWorldBinder>
{ }
```

HelloWorldData is an IAnimationJobData

```
public interface IAnimationJobData
{
    bool IsValid();
    void SetDefaultValues();
}
```

Contains necessary data to create the job

SCRIPTING A CUSTOM CONSTRAINT

```
public class HelloWorld : RigConstraint<HelloWorldJob, HelloWorldData, HelloWorldBinder>
{ }
```

HelloWorldData is an IAnimationJobData

```
[Serializable]
public struct HelloWorldData : IAnimationJobData
{
    public Transform constrainedObject;
    [SyncSceneToStream] public Transform sourceObject;

    public bool IsValid()
    {
        return !(constrainedObject == null || sourceObject == null);
    }

    public void SetDefaultValues()
    {
        constrainedObject = null;
        sourceObject = null;
    }
}
```

SCRIPTING A CUSTOM CONSTRAINT

```
public class HelloWorld : RigConstraint<HelloWorldJob, HelloWorldData, HelloWorldBinder>
{ }
```

HelloWorldBinder is an AnimationJobBinder

```
public abstract class AnimationJobBinder<TJob, TData> : IAnimationJobBinder
    where TJob : struct, IAnimationJob
    where TData : struct, IAnimationJobData
{
    public abstract TJob Create(Animator animator, ref TData data, Component component);

    public abstract void Destroy(TJob job);

    public virtual void Update(TJob job, ref TData data) {}
}
```

Create/Destroy/Update an IAnimationJob given some IAnimationJobData

SCRIPTING A CUSTOM CONSTRAINT

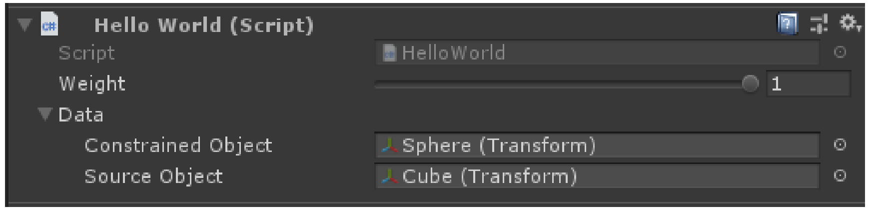
```
public class HelloWorld : RigConstraint<HelloWorldJob, HelloWorldData, HelloWorldBinder>
{ }
```

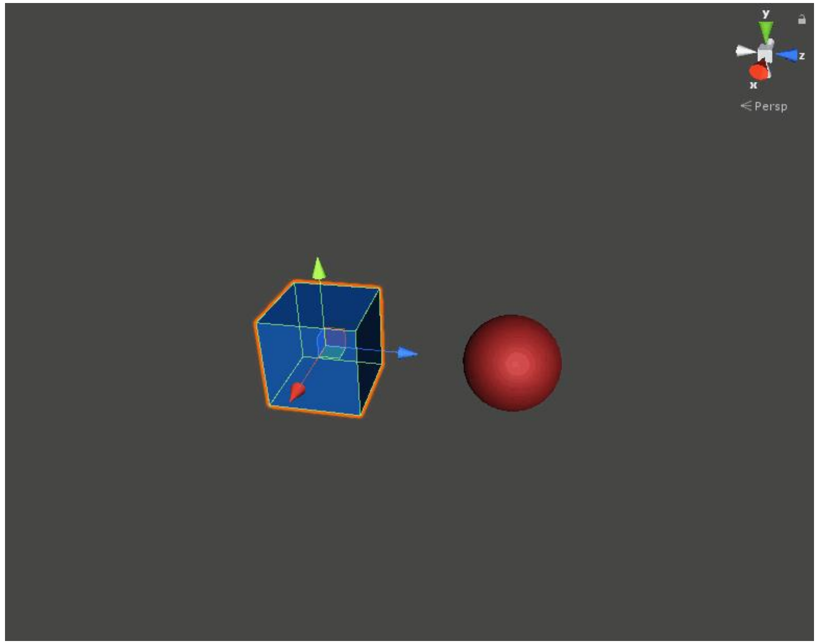
HelloWorldBinder is an AnimationJobBinder

```
public class HelloWorldBinder : AnimationJobBinder<HelloWorldJob, HelloWorldData>
{
    public override HelloWorldJob Create(Animator animator, ref HelloWorldData data, Component component)
    {
        return new HelloWorldJob()
        {
            constrained = ReadWriteTransformHandle.Bind(animator, data.constrainedObject),
            source = ReadOnlyTransformHandle.Bind(animator, data.sourceObject)
        };
    }

    public override void Destroy(HelloWorldJob job) { }
}
```

SCRIPTING A CUSTOM CONSTRAINT

```
public class HelloWorld : RigConstraint<HelloWorldJob, HelloWorldData, HelloWorldBinder>
{ }
```

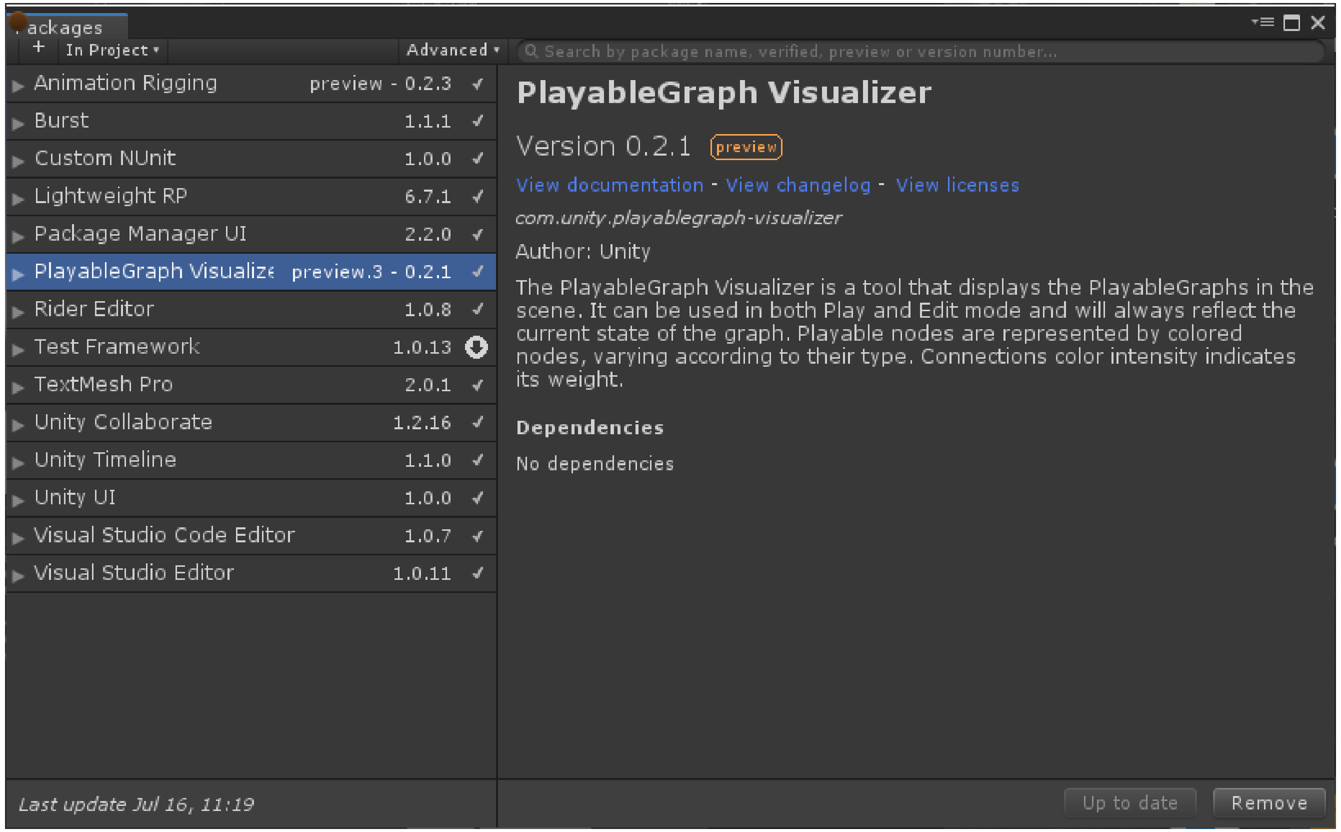


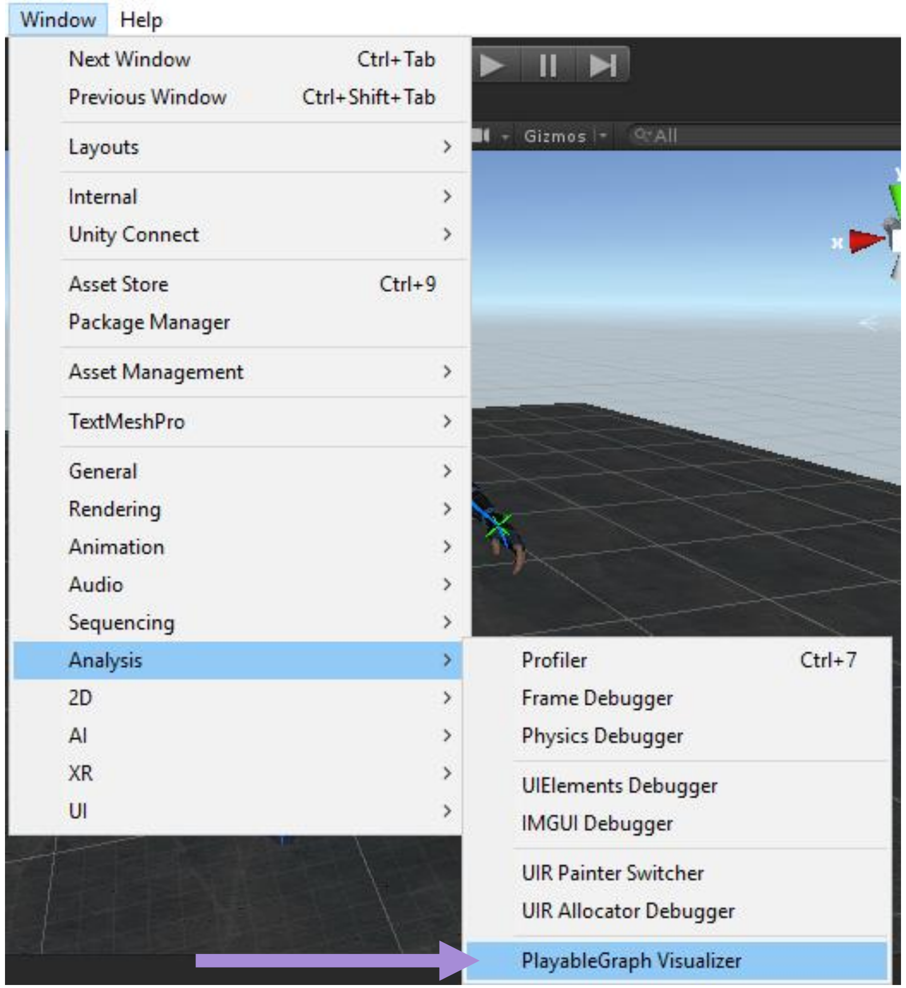


LIVE DEMO

- Lets rework the HelloWord constraint
- Open the **02_WorkshopCopyLocation** scene
- Add toggles on the constraint to invert the axis values

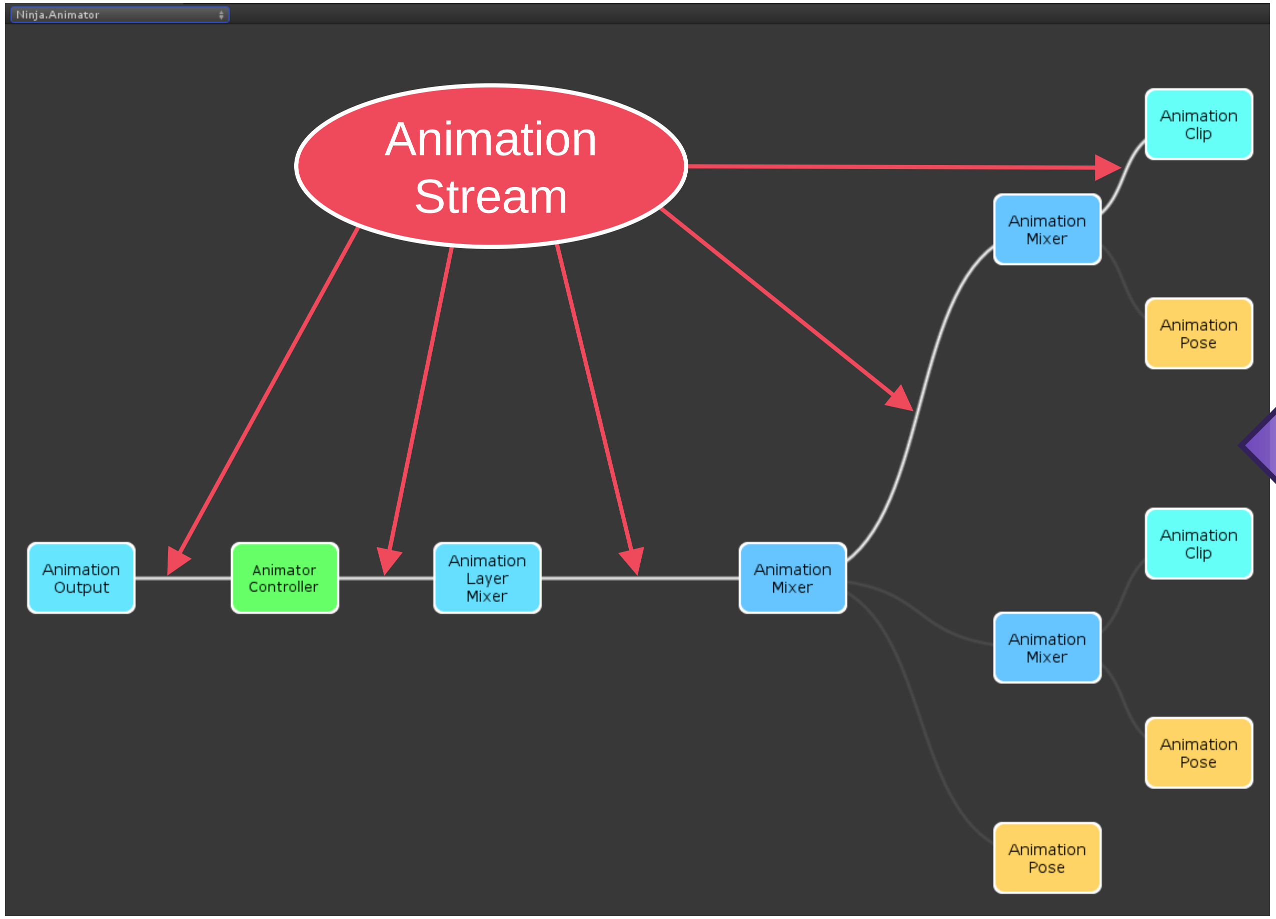
INSPECTING THE PLAYABLE GRAPH

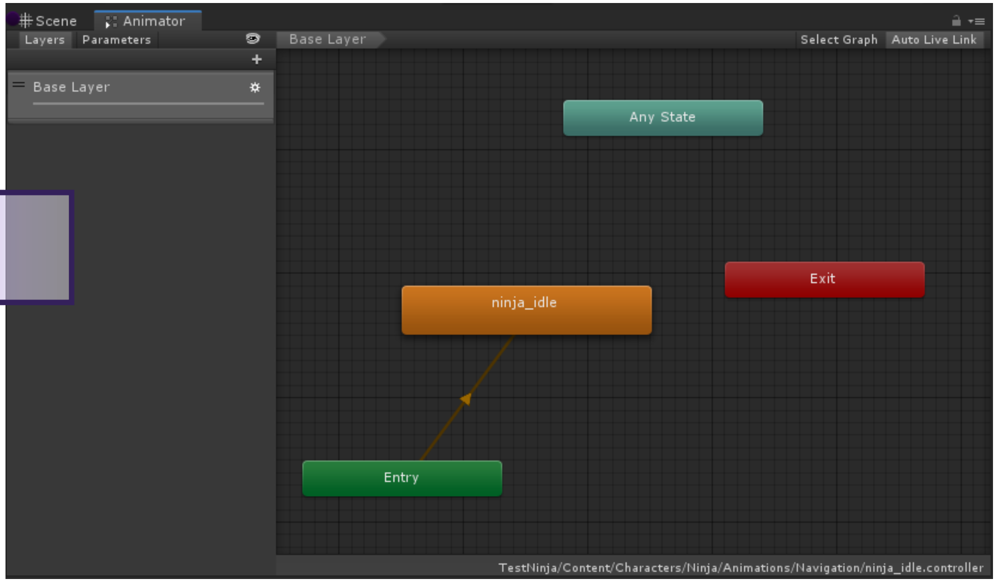




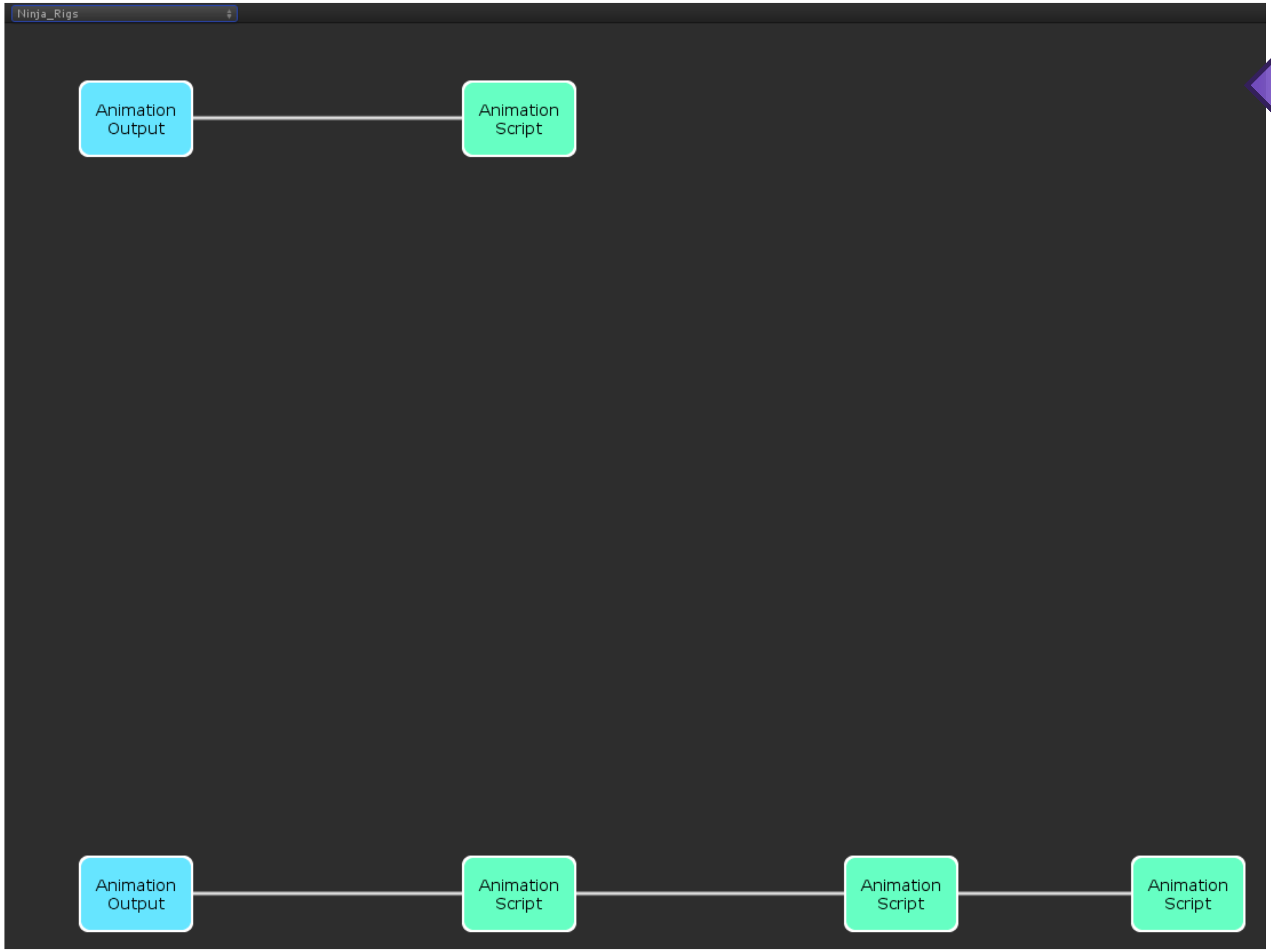
thrive SIGGRAPH2019 LOS ANGELES • 28 JULY – 1 AUGUST

INSPECTING THE PLAYABLE GRAPH

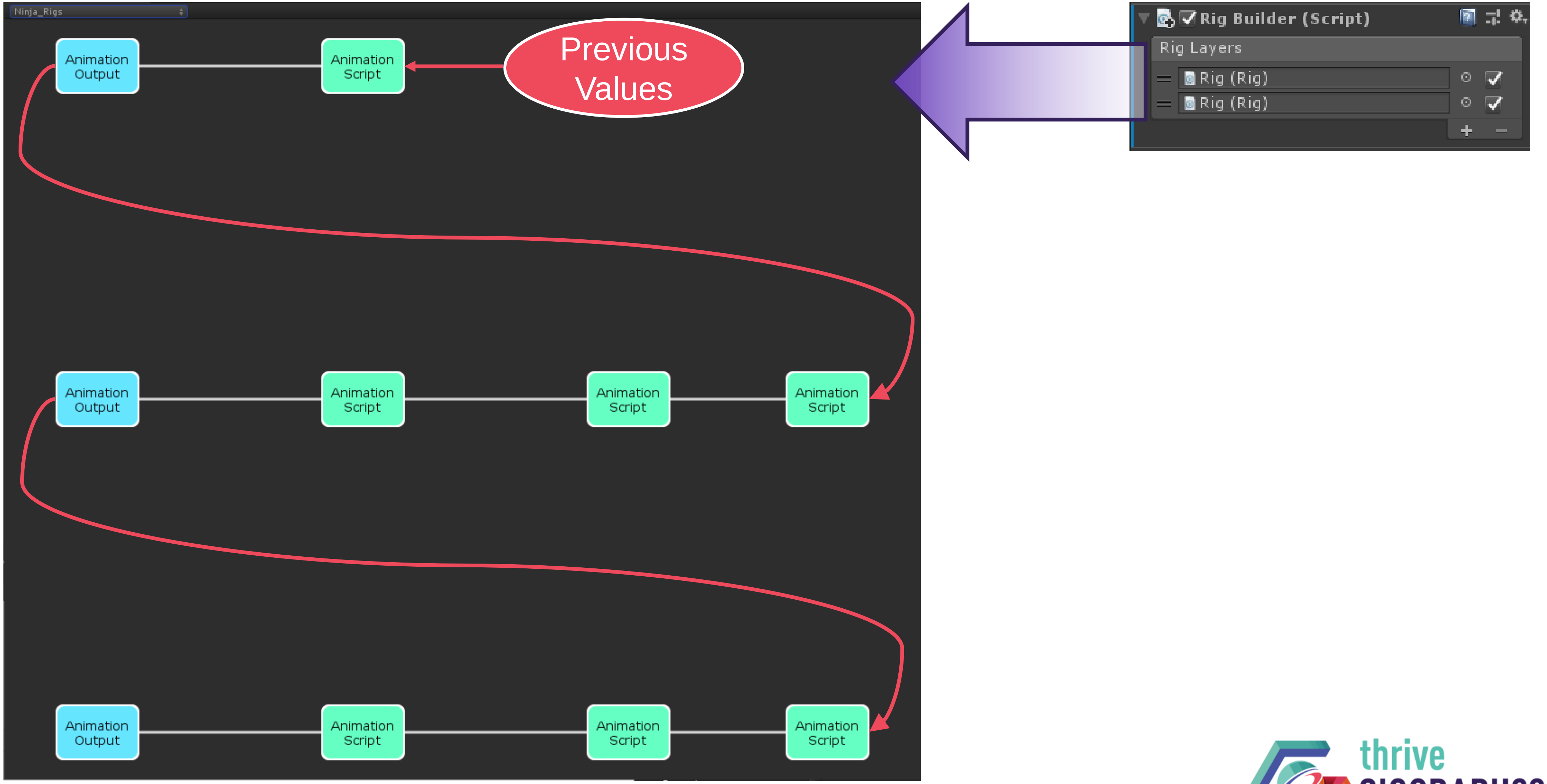




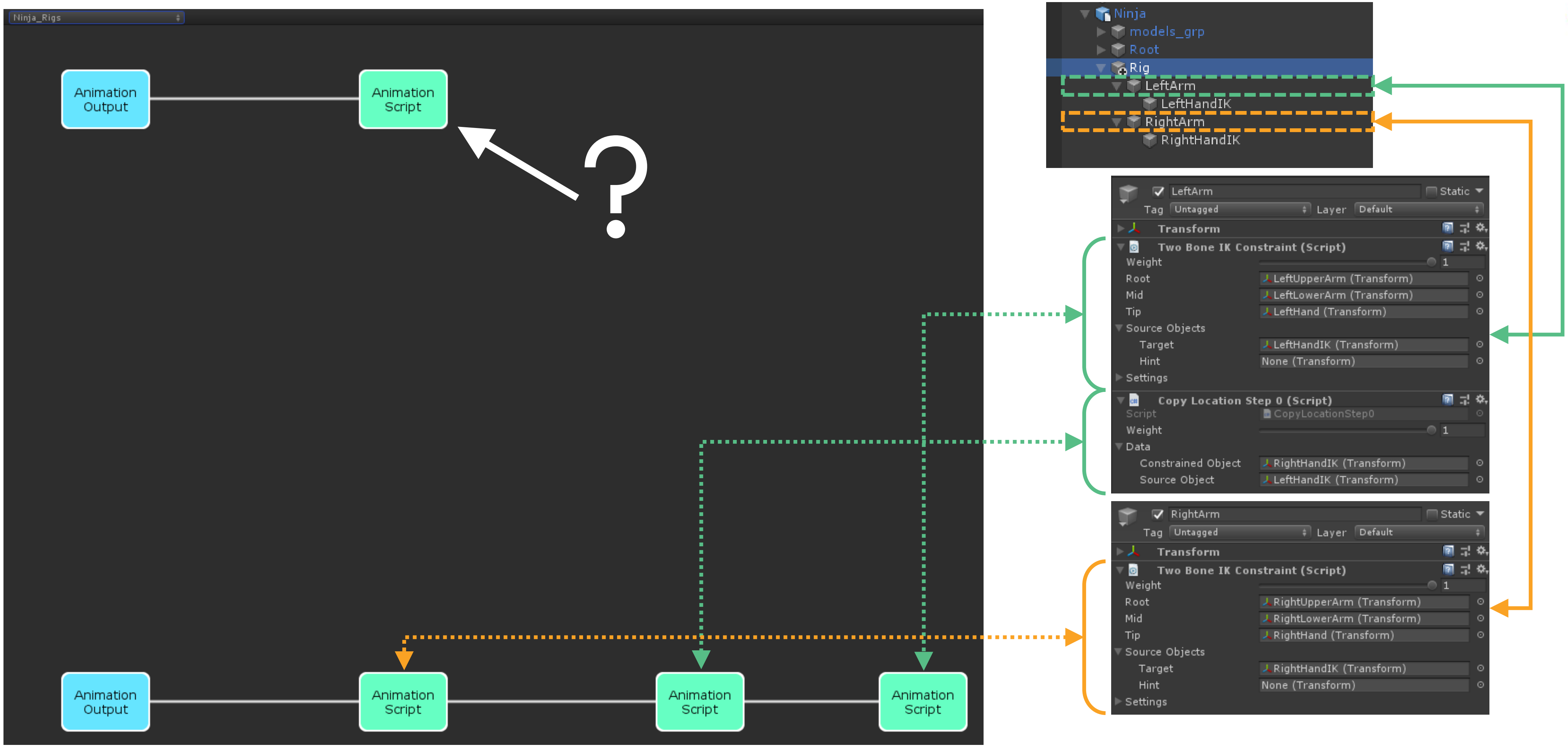
INSPECTING THE PLAYABLE GRAPH



INSPECTING THE PLAYABLE GRAPH



INSPECTING THE PLAYABLE GRAPH



SYNC VALUES TO ANIMATION STREAM

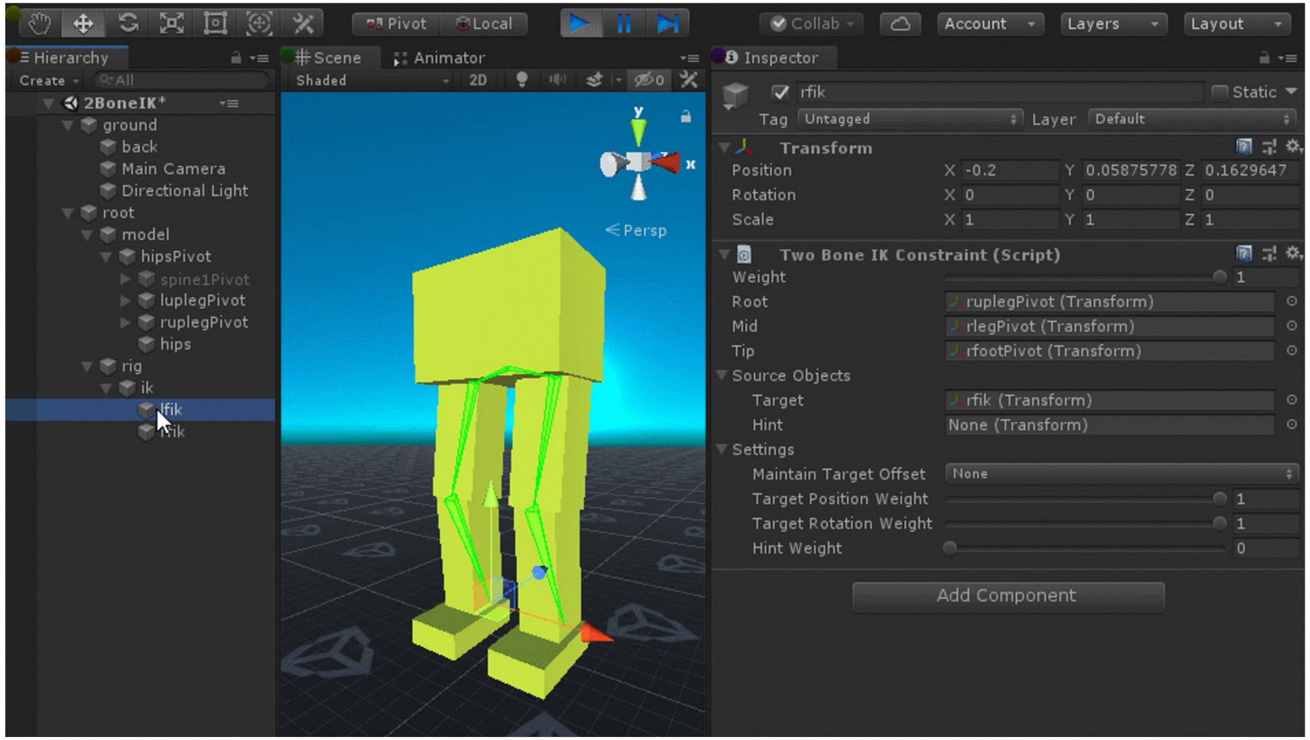
- First job to run before evaluating any of the rig layers

- Pushes latest scene values to the animation stream
  - Only if these have **NOT** previously been animated
  - Makes latest values available to all downstream jobs

- Flag constraint fields using **[SyncSceneToStream]**
  - Works on a limited set of data types:
    - Float, Int, Bool, Vector[2,3,4], Quaternions, Vector3Int, Vector3Bool
    - Transform, Transform[], WeightedTransform, WeightedTransformArray

SYNC VALUES TO ANIMATION STREAM

- RigTransform component
  - A quick way to sync transforms that are not referenced by any constraints



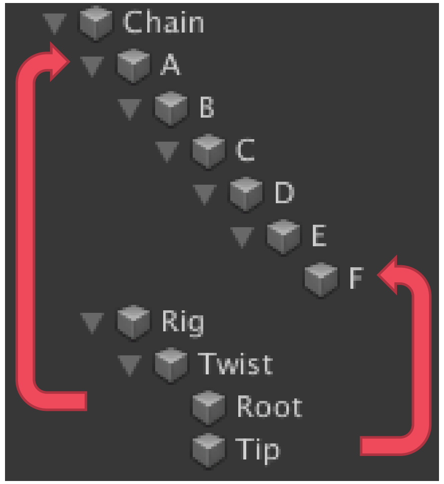
thrive SIGGRAPH2019 LOS ANGELES • 28 JULY – 1 AUGUST

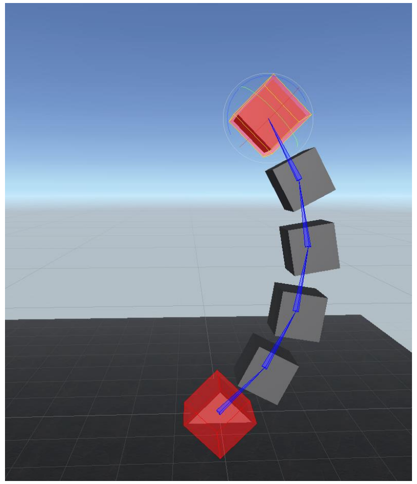
LIVE DEMO

- Let’s rework our CopyLocation constraint to make our newly added axis toggles dynamic and part of the AnimationStream.

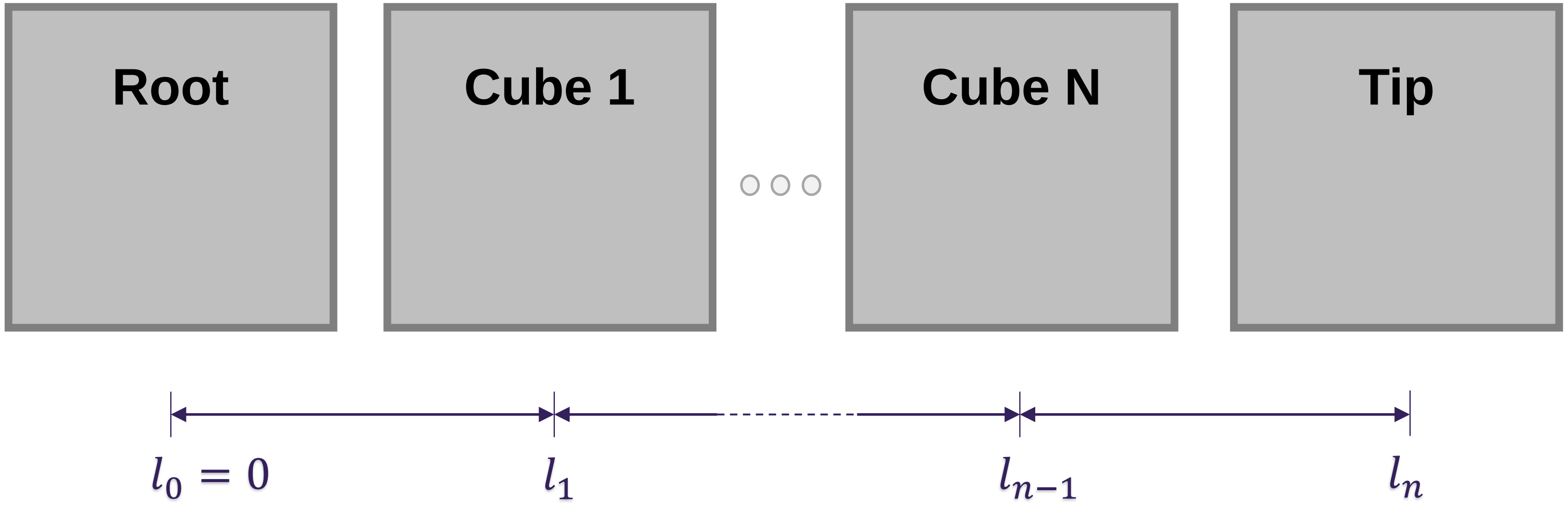
BUILDING A TWIST CHAIN CONSTRAINT

- We're proposing a bidirectional twist chain constraint driven by two effectors.
- We'll be limiting this constraint to animating rotation only.



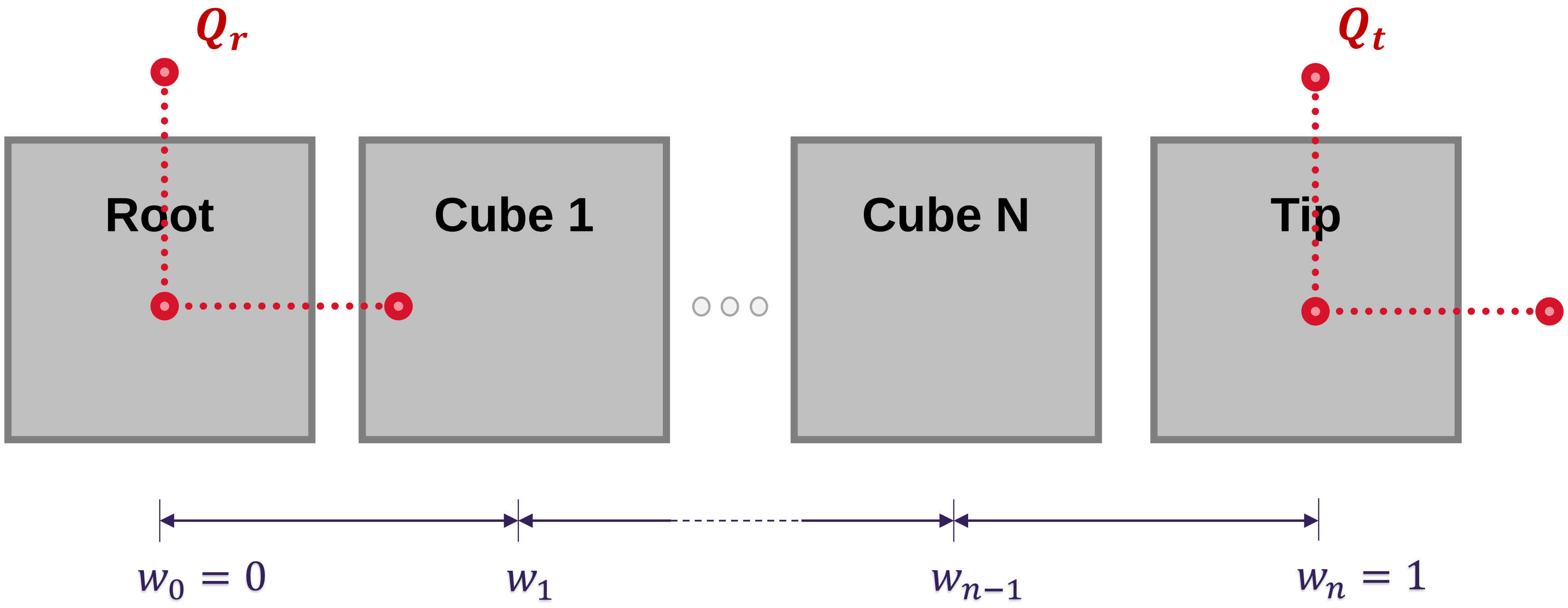


BUILDING A TWIST CHAIN CONSTRAINT



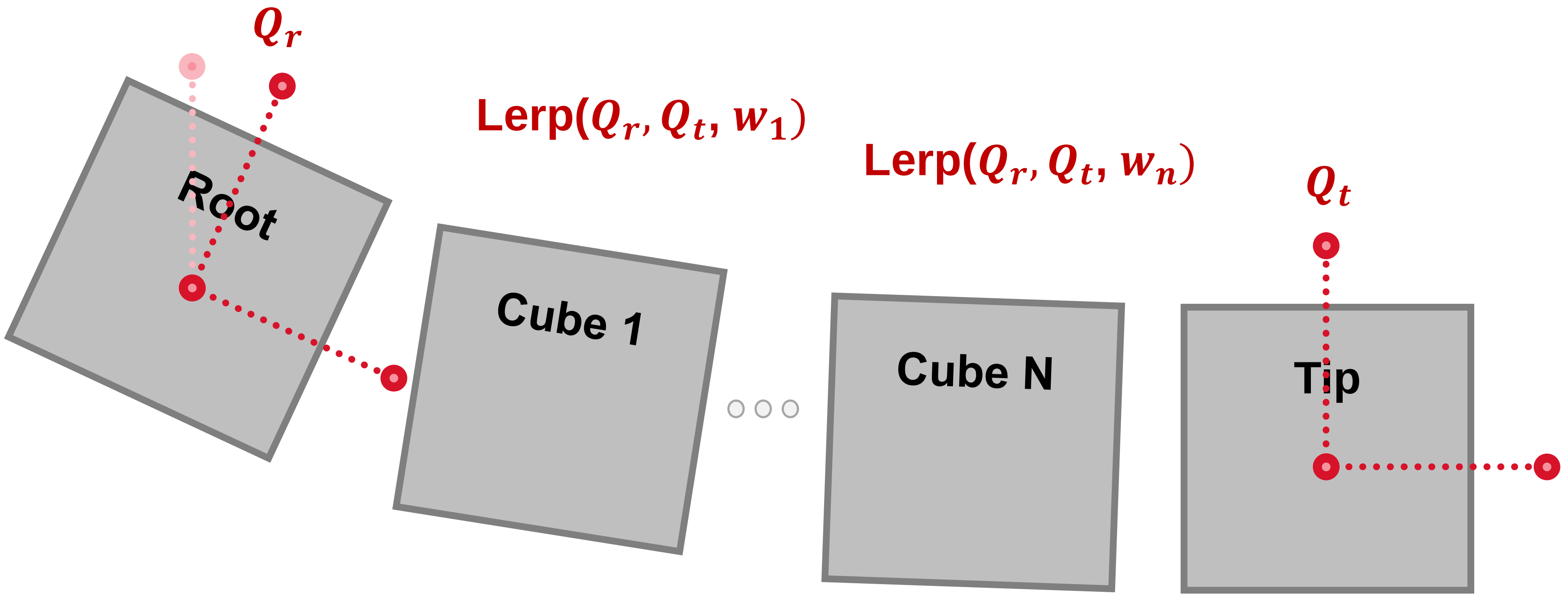
$$l_{[0,n]} \rightarrow w \in [0, 1]$$

BUILDING A TWIST CHAIN CONSTRAINT

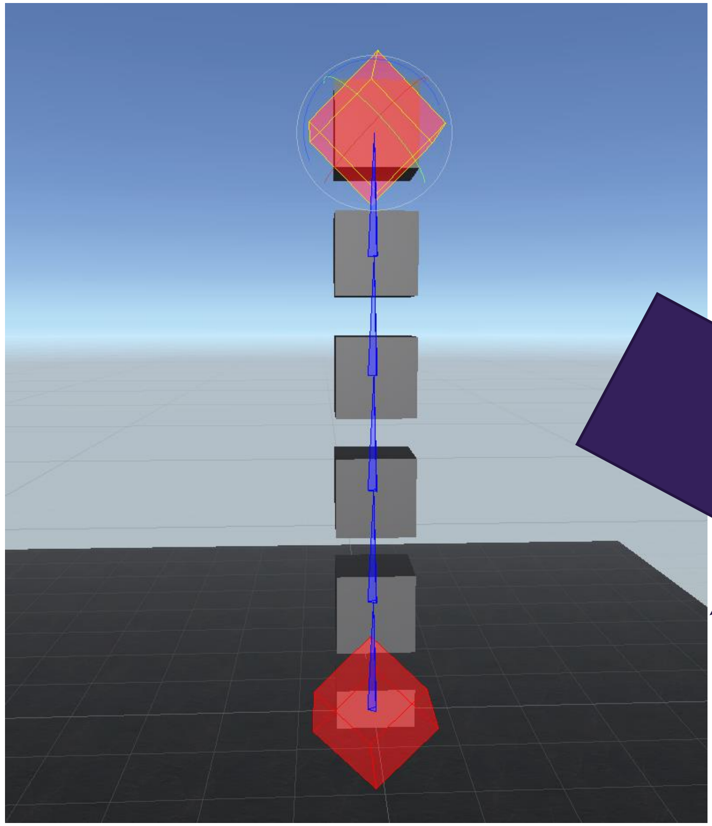


BUILDING A TWIST CHAIN CONSTRAINT

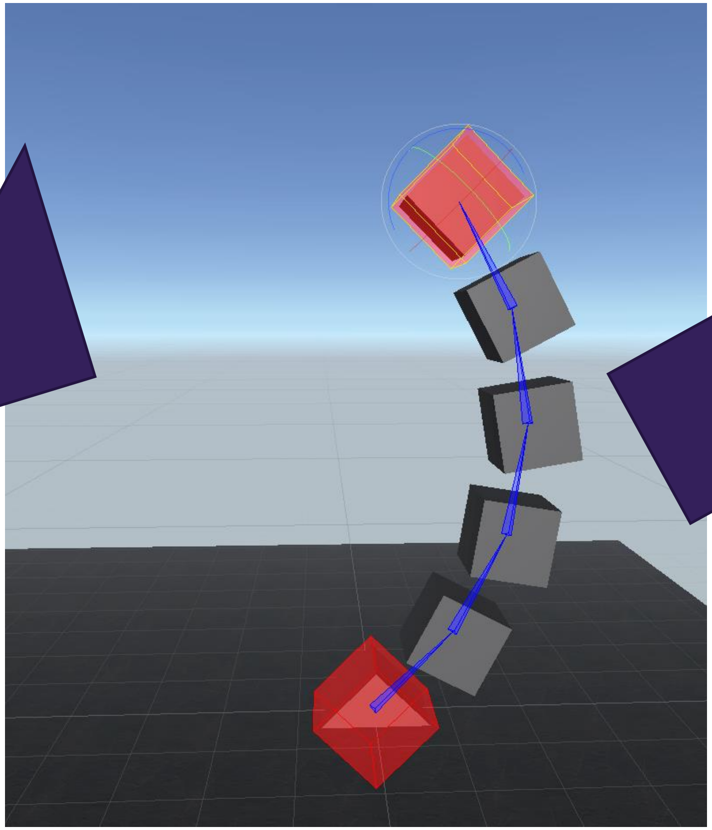
$w \in [0, 1]$



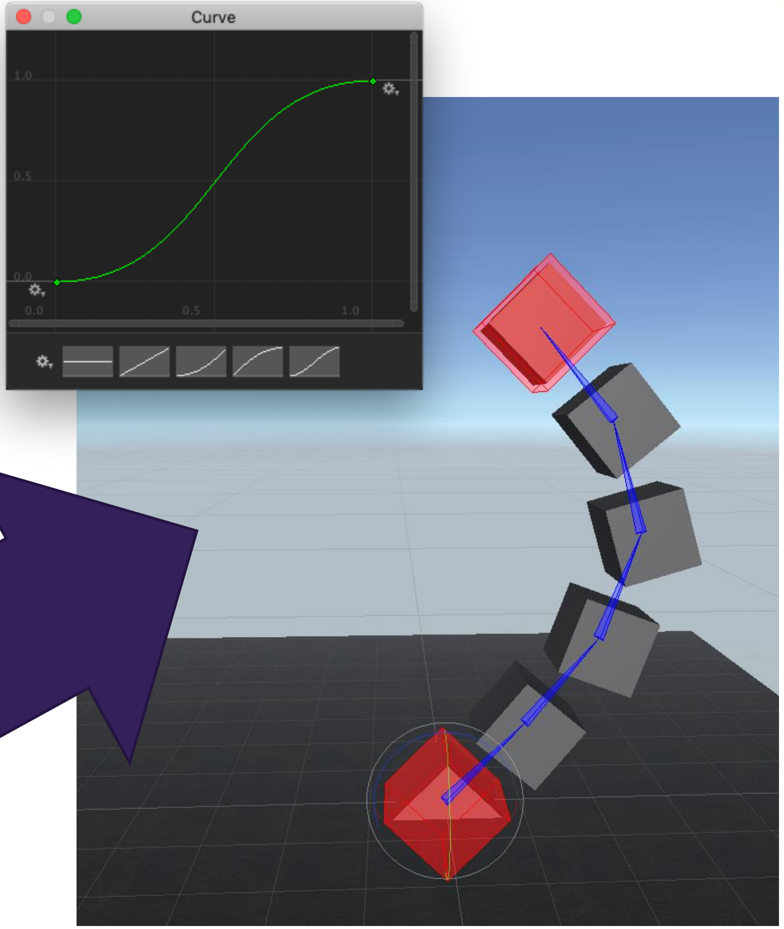
BUILDING A TWIST CHAIN CONSTRAINT



03_WorkshopTwistChain &
04_WorkshopTwistChain



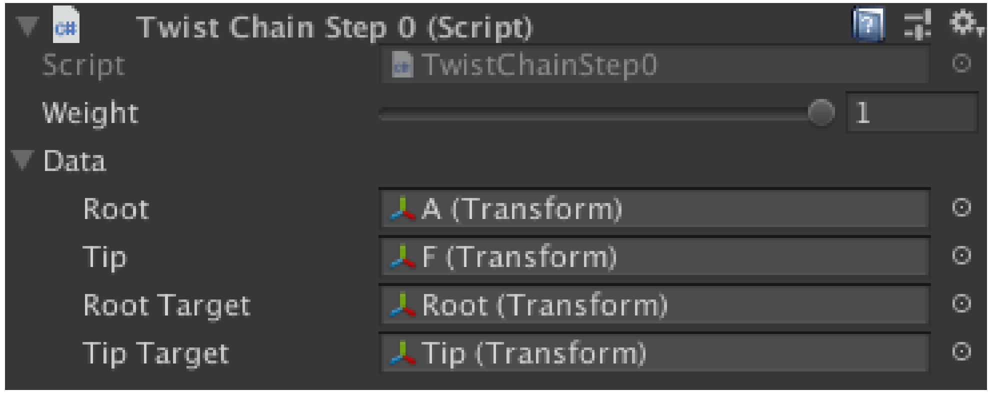
05_WorkshopTwistChain

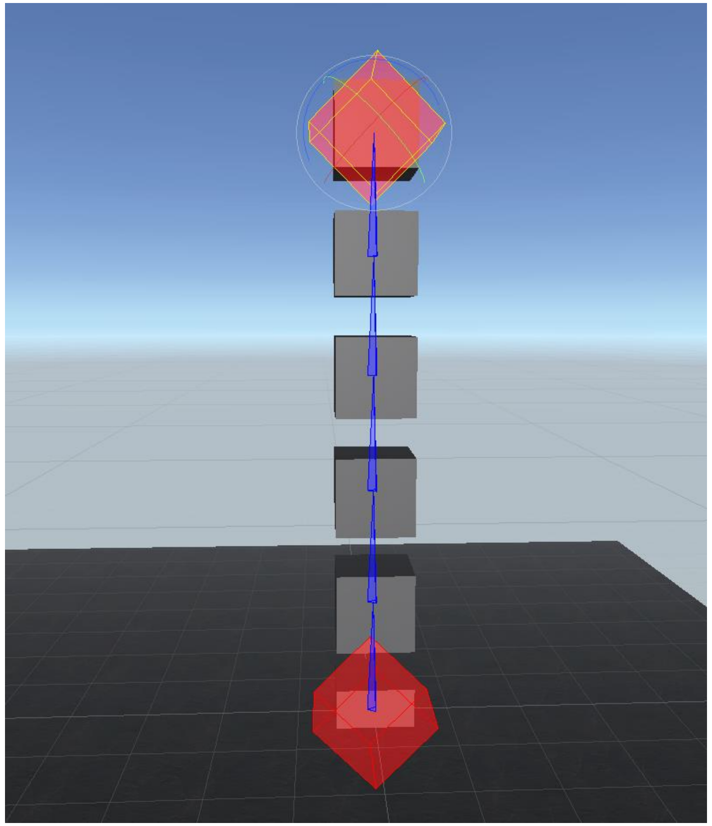


Final_WorkshopTwistChain

BUILDING A TWIST CHAIN CONSTRAINT

- Open scene 03_WorkshopTwistChain.
- The scene is already setup with a basic hierarchy and RigBuilder.
- The constraint TwistChainStep0 will execute, but does nothing at the moment.





TwistChainStep0.unity

BUILDING A TWIST CHAIN CONSTRAINT

- TwistChainData

```
[System.Serializable]
public struct TwistChainStep0Data : IAnimationJobData
{
    public Transform root;
    public Transform tip;

    [SyncSceneToStream] public Transform rootTarget;
    [SyncSceneToStream] public Transform tipTarget;

    bool IAnimationJobData.IsValid() => !(root == null || tip == null || !tip.IsChildOf(root) || rootTarget == null || tipTarget == null);

    void IAnimationJobData.SetDefaultValues()
    {
        root = tip = rootTarget = tipTarget = null;
    }
}
```

BUILDING A TWIST CHAIN CONSTRAINT

- TwistChainData

```
[System.Serializable]
public struct TwistChainStep0Data : IAnimationJobData
{
    public Transform root;
    public Transform tip;

    [SyncSceneToStream] public Transform rootTarget;
    [SyncSceneToStream] public Transform tipTarget;

    bool IAnimationJobData.IsValid() => !(root == null || tip == null || !tip.IsChildOf(root) || rootTarget == null || tipTarget == null);

    void IAnimationJobData.SetDefaultValues()
    {
        root = tip = rootTarget = tipTarget = null;
    }
}
```

BUILDING A TWIST CHAIN CONSTRAINT

- TwistChainData

```
[System.Serializable]
public struct TwistChainStep0Data : IAnimationJobData
{
    public Transform root;
    public Transform tip;

    [SyncSceneToStream] public Transform rootTarget;
    [SyncSceneToStream] public Transform tipTarget;

    bool IAnimationJobData.IsValid() => !(root == null || tip == null || !tip.IsChildOf(root) || rootTarget == null || tipTarget == null);

    void IAnimationJobData.SetDefaultValues()
    {
        root = tip = rootTarget = tipTarget = null;
    }
}
```

BUILDING A TWIST CHAIN CONSTRAINT

- TwistChainData

```csharp
[System.Serializable]
public struct TwistChainStep0Data : IAnimationJobData
{
    public Transform root;
    public Transform tip;

    [SyncSceneToStream] public Transform rootTarget;
    [SyncSceneToStream] public Transform tipTarget;

    bool IAnimationJobData.IsValid() => !(root == null || tip == null || !tip.IsChildOf(root) || rootTarget == null || tipTarget == null);

    void IAnimationJobData.SetDefaultValues()
    {
        root = tip = rootTarget = tipTarget = null;
    }
}
```

BUILDING A TWIST CHAIN CONSTRAINT

- TwistChainData

```
[System.Serializable]
public struct TwistChainStep0Data : IAnimationJobData
{
    public Transform root;
    public Transform tip;

    [SyncSceneToStream] public Transform rootTarget;
    [SyncSceneToStream] public Transform tipTarget;

    bool IAnimationJobData.IsValid() => !(root == null || tip == null || !tip.IsChildOf(root) || rootTarget == null || tipTarget == null);

    void IAnimationJobData.SetDefaultValues()
    {
        root = tip = rootTarget = tipTarget = null;
    }
}
```

BUILDING A TWIST CHAIN CONSTRAINT

- TwistChainJob

```
[Unity.Burst.BurstCompile]
public struct TwistChainStep0Job : IWeightedAnimationJob
{
    public ReadWriteTransformHandle rootTarget;
    public ReadWriteTransformHandle tipTarget;

    public NativeArray<ReadWriteTransformHandle> chain;

    public NativeArray<float> steps;

    public FloatProperty jobWeight { get; set; }

    public void ProcessRootMotion(AnimationStream stream) {}

    public void ProcessAnimation(AnimationStream stream) {}
}
```

BUILDING A TWIST CHAIN CONSTRAINT

- TwistChainJob

```
[Unity.Burst.BurstCompile]
public struct TwistChainStep0Job : IWeightedAnimationJob
{
    public ReadWriteTransformHandle rootTarget;
    public ReadWriteTransformHandle tipTarget;

    public NativeArray<ReadWriteTransformHandle> chain;

    public NativeArray<float> steps;

    public FloatProperty jobWeight { get; set; }

    public void ProcessRootMotion(AnimationStream stream) {}

    public void ProcessAnimation(AnimationStream stream) {}
}
```

BUILDING A TWIST CHAIN CONSTRAINT

- TwistChainJob

```
[Unity.Burst.BurstCompile]
public struct TwistChainStep0Job : IWeightedAnimationJob
{
    public ReadWriteTransformHandle rootTarget;
    public ReadWriteTransformHandle tipTarget;

    public NativeArray<ReadWriteTransformHandle> chain;

    public NativeArray<float> steps;

    public FloatProperty jobWeight { get; set; }

    public void ProcessRootMotion(AnimationStream stream) {}

    public void ProcessAnimation(AnimationStream stream) {}
}
```

BUILDING A TWIST CHAIN CONSTRAINT

- TwistChainJob

```
[Unity.Burst.BurstCompile]
public struct TwistChainStep0Job : IWeightedAnimationJob
{
    public ReadWriteTransformHandle rootTarget;
    public ReadWriteTransformHandle tipTarget;

    public NativeArray<ReadWriteTransformHandle> chain;

    public NativeArray<float> steps;

    public FloatProperty jobWeight { get; set; }

    public void ProcessRootMotion(AnimationStream stream) {}

    public void ProcessAnimation(AnimationStream stream) {}
}
```

BUILDING A TWIST CHAIN CONSTRAINT

- TwistChainJob

```
[Unity.Burst.BurstCompile]
public struct TwistChainStep0Job : IWeightedAnimationJob
{
    public ReadWriteTransformHandle rootTarget;
    public ReadWriteTransformHandle tipTarget;

    public NativeArray<ReadWriteTransformHandle> chain;

    public NativeArray<float> steps;

    public FloatProperty jobWeight { get; set; }

    public void ProcessRootMotion(AnimationStream stream) {}

    public void ProcessAnimation(AnimationStream stream) {}
}
```


thrive
SIGGRAPH2019
LOS ANGELES • 28 JULY – 1 AUGUST

BUILDING A TWIST CHAIN CONSTRAINT

- TwistChainJobBinder

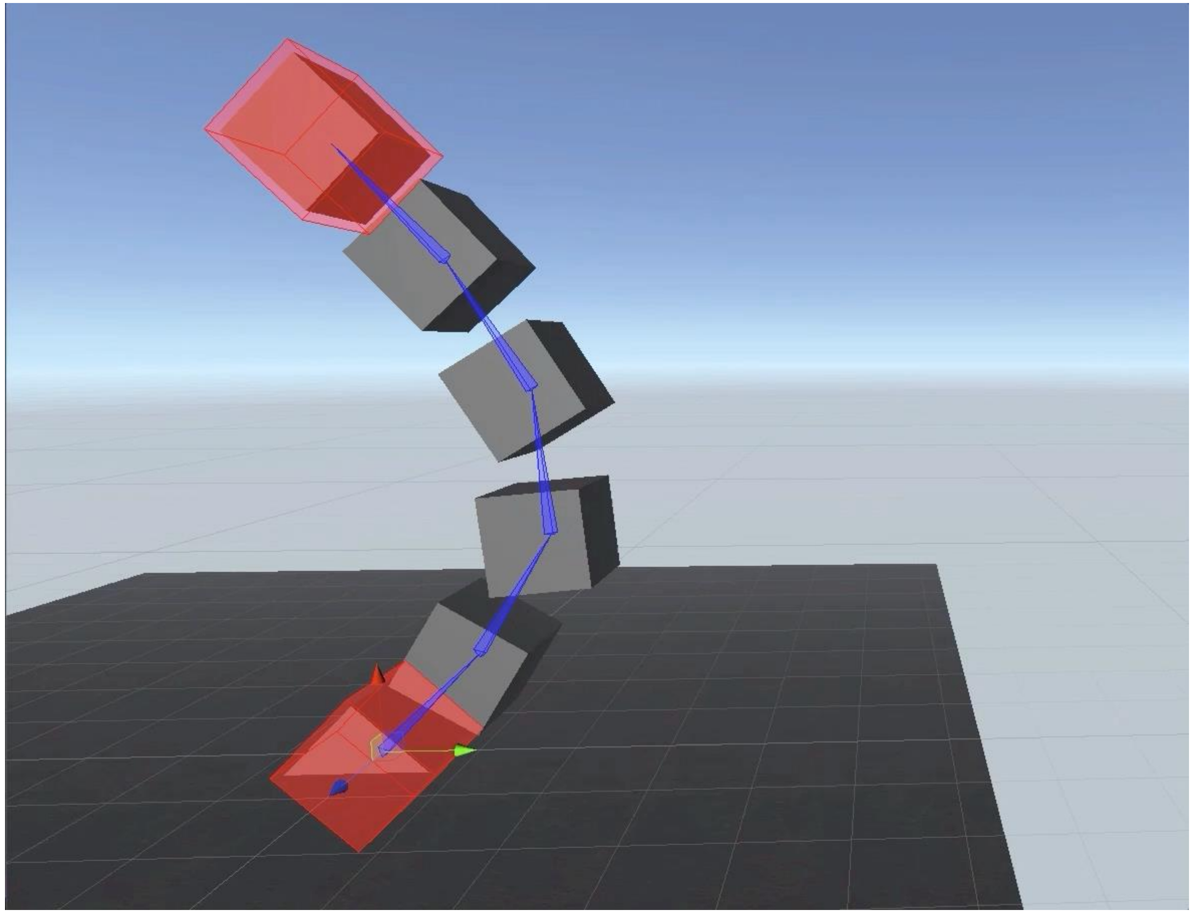
```
public class TwistChainStep0JobBinder : AnimationJobBinder<TwistChainStep0Job, TwistChainStep0Data>
{
    public override TwistChainStep0Job Create(Animator animator, ref TwistChainStep0Data data, Component component)
    {
        // Build Job.
        var job = new TwistChainStep0Job();
        return job;
    }

    public override void Destroy(TwistChainStep0Job job)
    {
    }

    public override void Update(TwistChainStep0Job job, ref TwistChainStep0Data data)
    {
    }
}
```

BUILDING A TWIST CHAIN CONSTRAINT

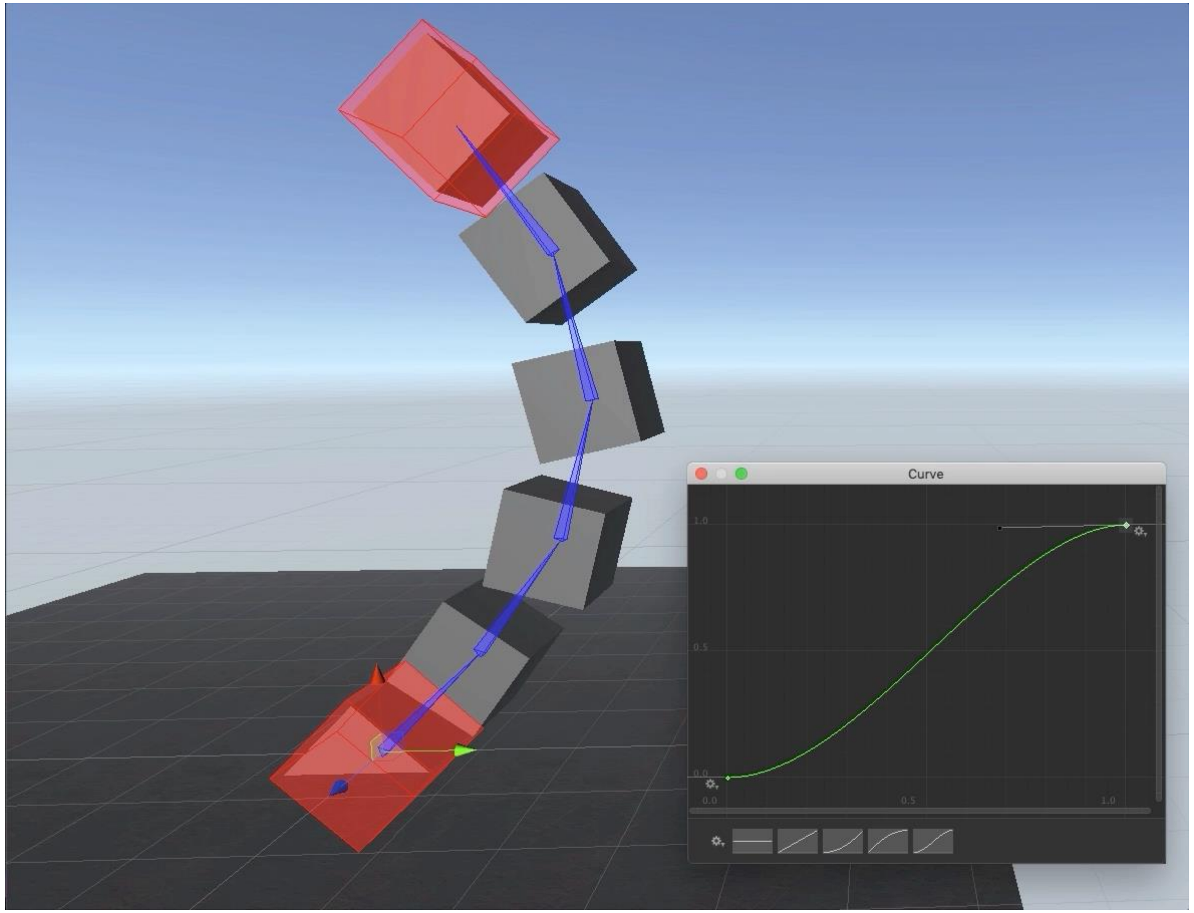
- Finishing up TwistChainJob

```
ReadWriteTransformHandle.GetPosition(AnimationStream stream);
ReadWriteTransformHandle.SetPosition(AnimationStream stream, Vector3 pos);

ReadWriteTransformHandle.GetRotation(AnimationStream stream);
ReadWriteTransformHandle.SetRotation(AnimationStream stream, Quaternion quat);

FloatProperty.Get(AnimationStream stream);
FloatProperty.Set(AnimationStream stream, float value);

Quaternion.Lerp(Quaternion a, Quaternion b, float t);
```

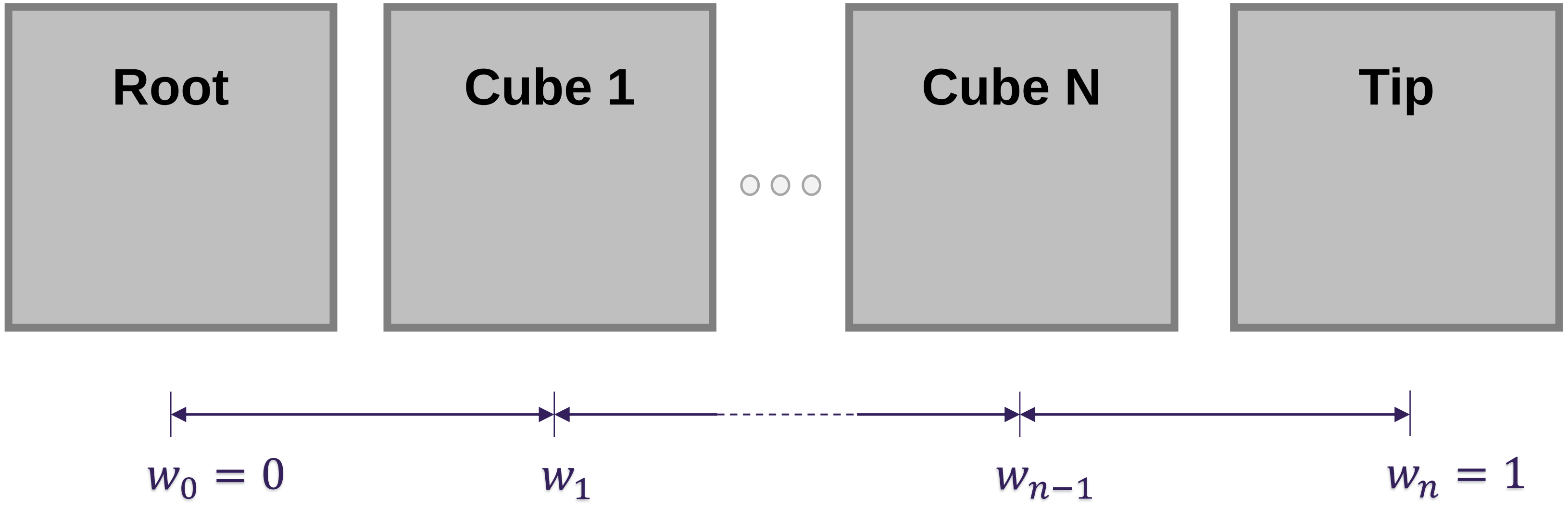
BUILDING A TWIST CHAIN CONSTRAINT



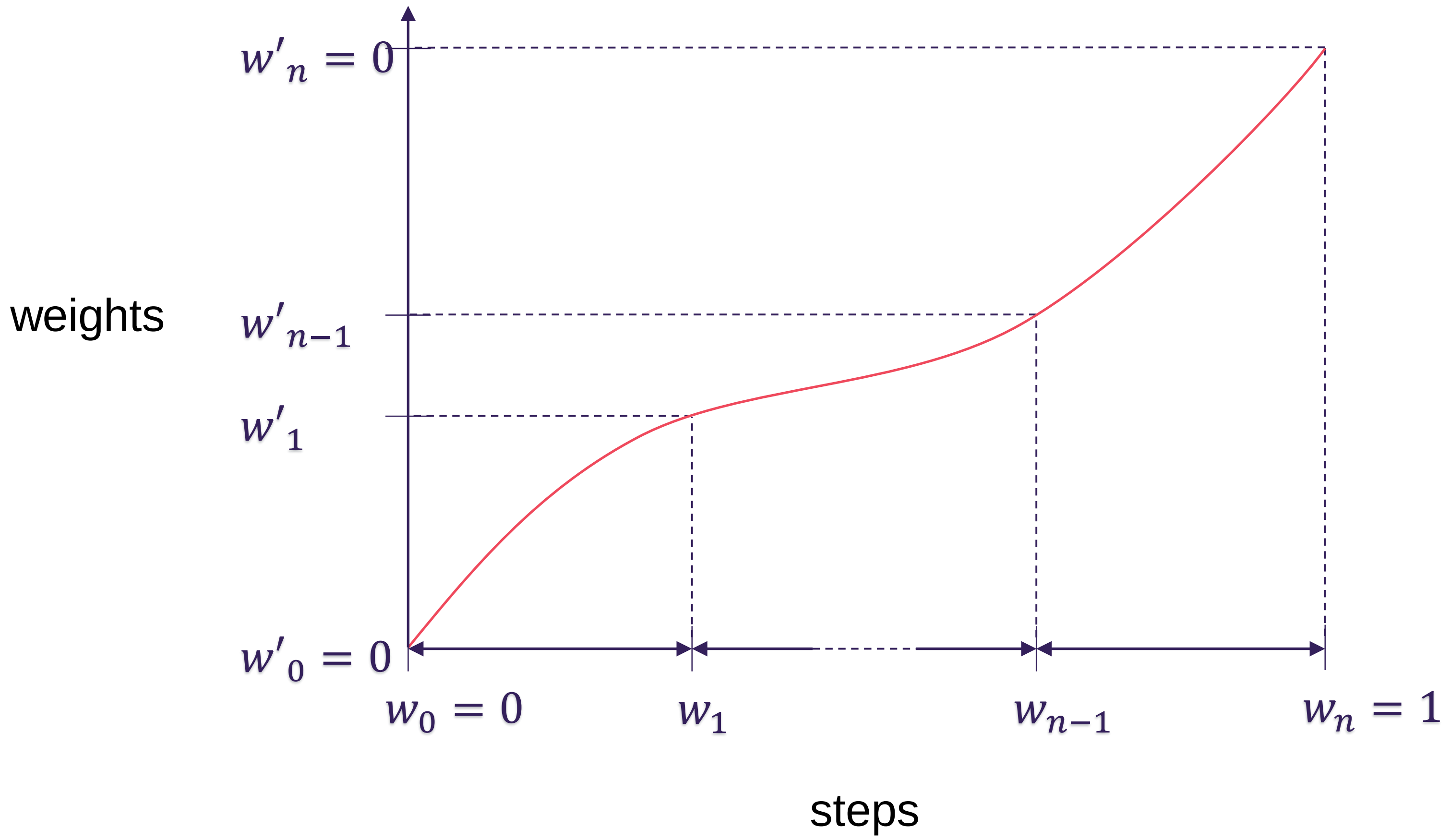
IMPROVING THE TWIST CHAIN CONSTRAINT



IMPROVING THE TWIST CHAIN CONSTRAINT

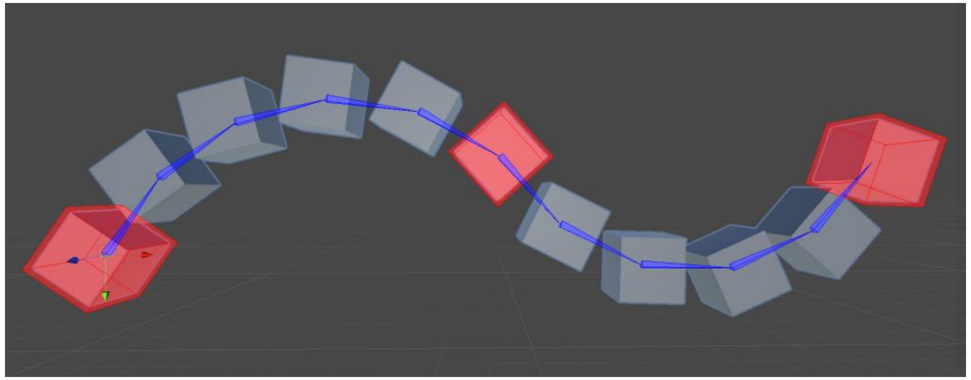


IMPROVING THE TWIST CHAIN CONSTRAINT

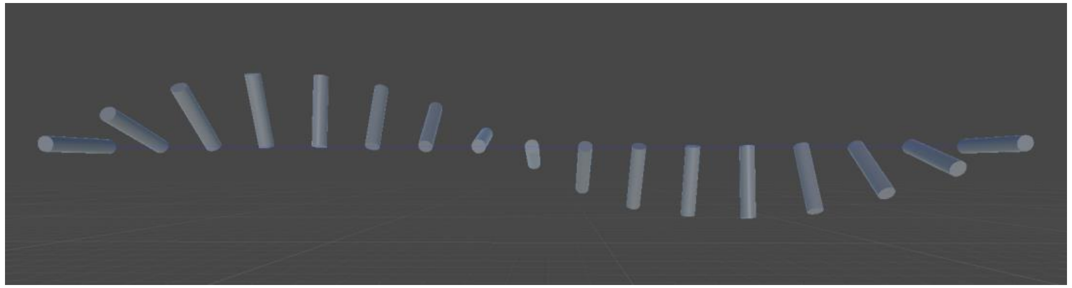


IMPROVING THE TWIST CHAIN CONSTRAINT

- Customizing further…
  - Animate weight parameters for more dynamic results.
  - Additional twist effectors for more control.



  - Euler twist effectors for spins above 180 degrees.



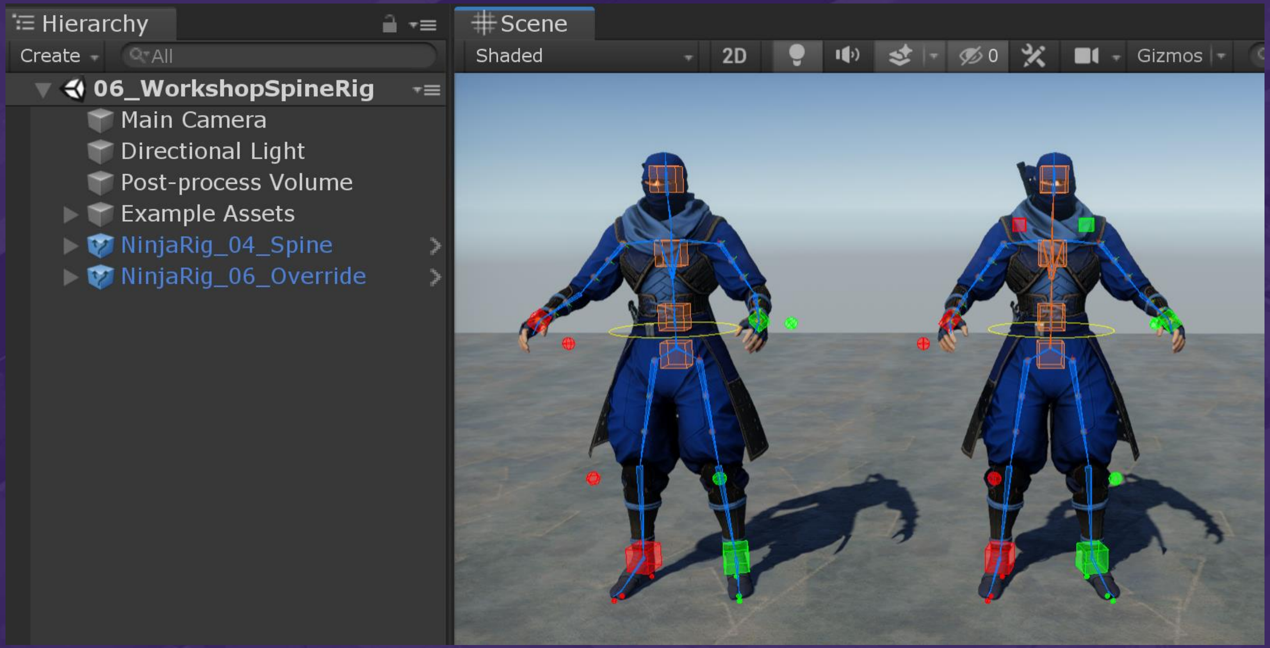
thrive
SIGGRAPH2019
LOS ANGELES • 28 JULY – 1 AUGUST

thrive SIGGRAPH2019
LOS ANGELES • 28 JULY – 1 AUGUST

WRAPPING UP

Setting up the spine region

- Open the scene: 06_WorkshopSpineRig



SETTING UP THE SPINE REGION – COG AND PELVIS

- Spine region – **Setup instructions**
  1. Add a new GameObject for the Spine region (child of ControlRig)

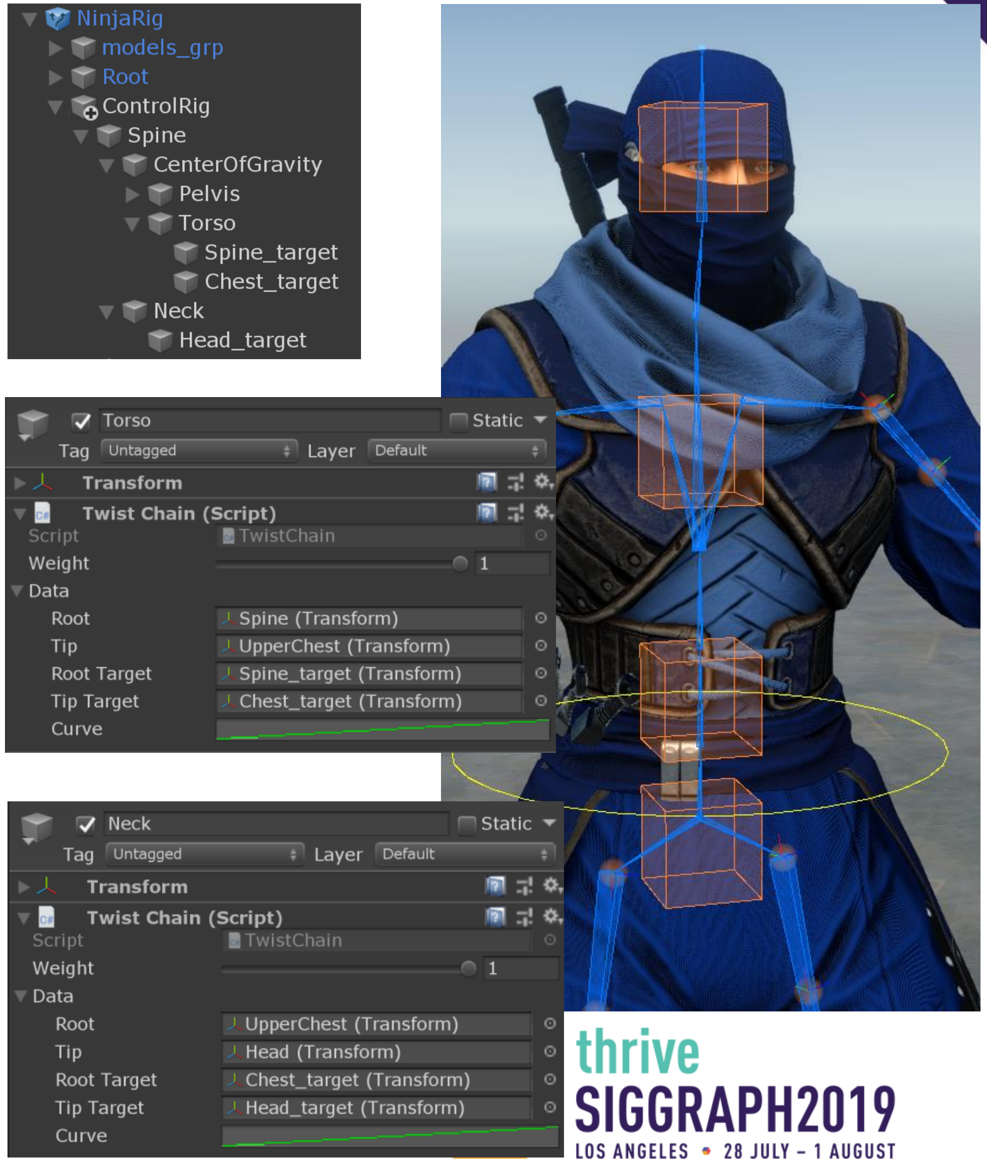
- Center of gravity – **Setup instructions**
  1. Add a child GameObject named CenterOfGravity (child of Spine)
  2. Add a Rig Transform component (so that it acts as the parent)

- Pelvis – **Setup instructions**
  1. Add a new GameObject named Pelvis (child of CenterOfGravity)
  2. Add a child GameObject named Pelvis_parentConstraint
     1. Add a Multi-Parent Constraint
        1. Constrained Object: Hips bone, Source Object: Pelvis_parentConstraint
        2. In Settings, set Maintain Offset to None
     2. Add a Multi-Referential Constraint and assign itself as the first Reference Object
  3. Add a new GameObject named Pelvis_target (child of Pelvis)
     1. Align this to the first Spine bone using: Animation Rigging > Align Transform
     2. Customize the Effector display options
     3. Assign this as a Reference Object of the Multi-Referential Constraint and set it to be the Driving object

SETTING UP THE SPINE REGION – TORSO AND HEAD

- Torso
  - **Setup instructions**
    1. Create a new GameObject for the Torso (child of CenterOfGravity)
    2. Add a TwistChain constraint
    3. Create two child GameObjects for Spine_target and Chest_target
       1. Align them to the Spine and UpperChest bones
       2. Customize the Effector display options
       3. Assign the TwistChain Root: Spine_target and Tip: Chest_target

- Neck and Head
  - **Setup instructions**
    1. Create a new GameObject for the Neck (child of CenterOfGravity)
    2. Add a TwistChain constraint
    3. Create a child GameObject for the Head
       1. Align it to the Head bone
       2. Customize the Effector display options
       3. Assign the TwistChain Root: Chest_target and Tip: Head_target

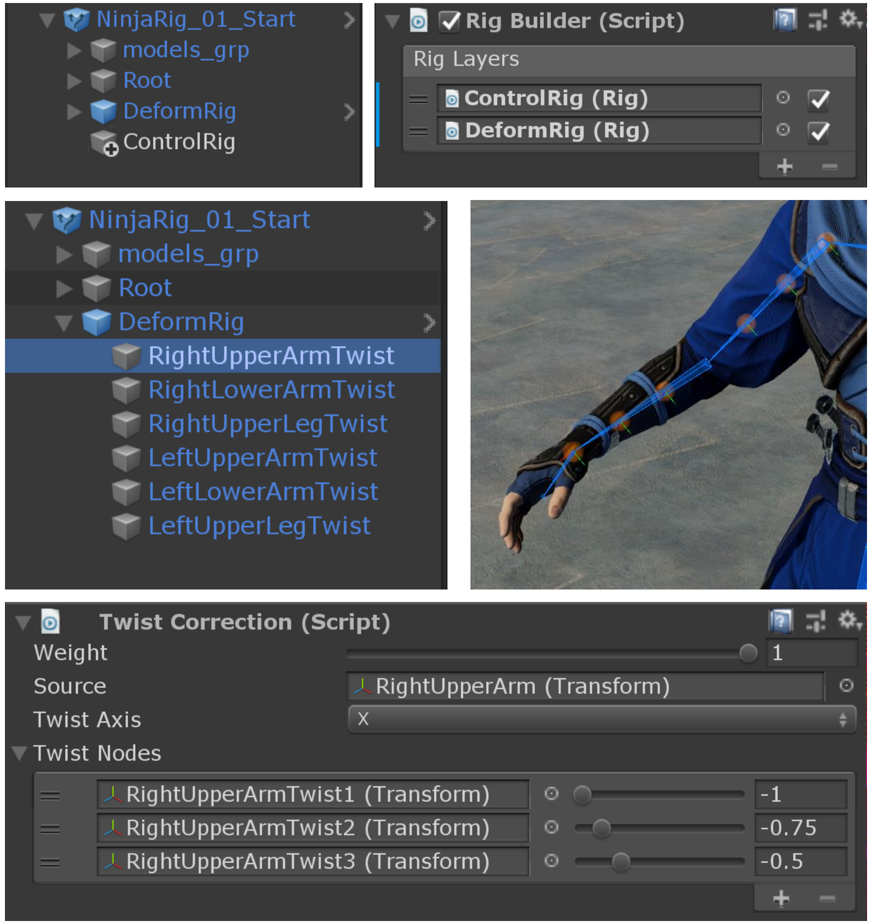


SETTING UP THE DEFORM RIG

- Deform Rig
  - Deform rig = bones that move automatically in relation to other bones in the skeleton
  - Used to create better looking deformations

- Twist Correction constraints
  - On the complete ninja rig observe the DeformRig
  - Assets/AnimationRiggingWorkshop/NinjaRigPrefabs/NinjaRig_05_Complete
    - Twist Correction constraints are setup for arms and legs
    - Select RightUpperArmTwist to see the constraint Inspector

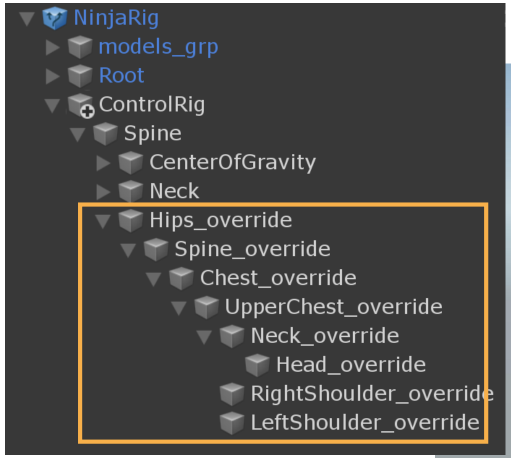
- Note: only one instance of a given rig constraint is allowed per GameObject – this is so that they can be animated

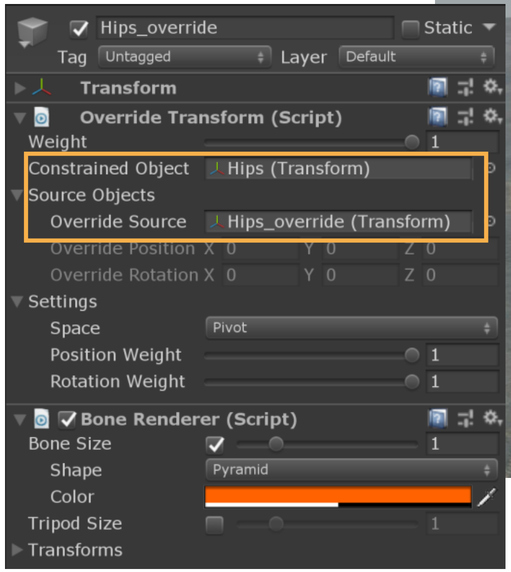


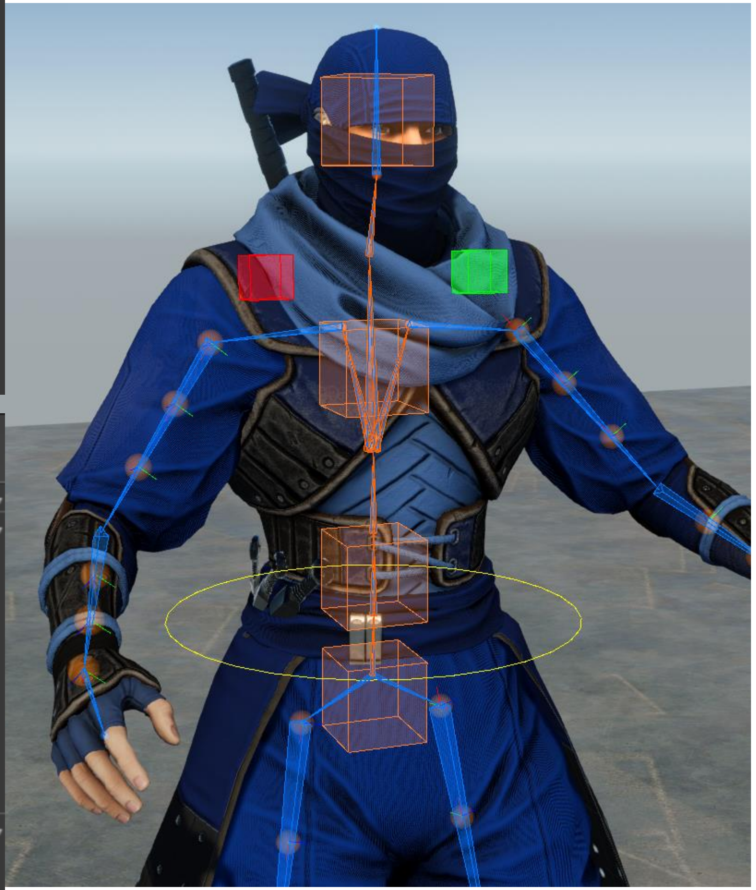
SETTING UP A SPINE OVERRIDE RIG

- Spine override rig
  - Enables animation to play with an offset. (can be static or animated)
  - Example: NinjaRig_06_Override

  - **Setup instructions**
    - Create a duplicate hierarchy for the spine
      - Hips → Hips_override, etc.
      - Add a Bone Renderer to the override bones
    - Set all rig constraints drive the override bones
    - Add Override Transform constraints to the override bones and set them to drive the actual spine bones



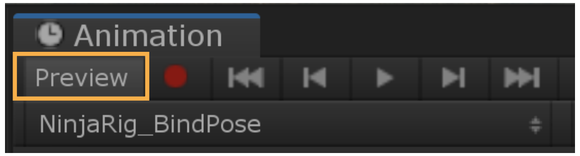


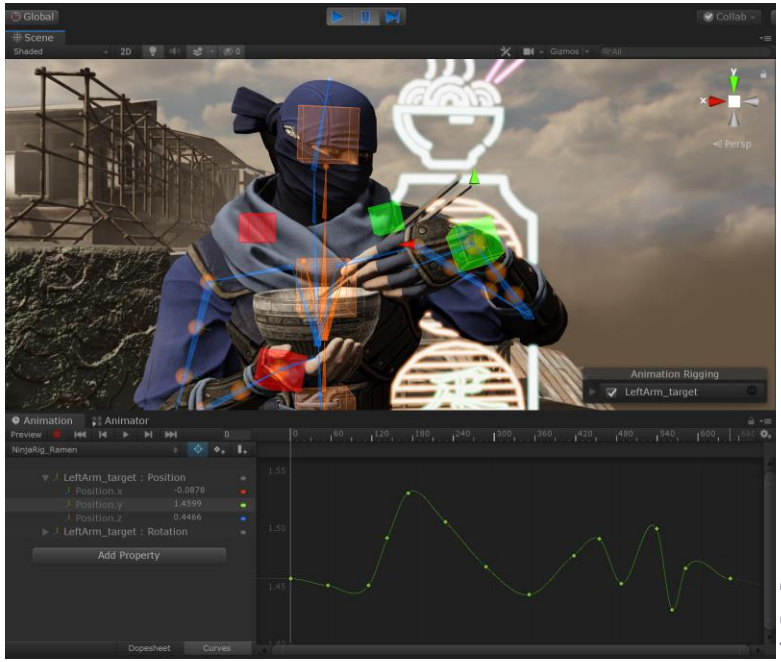


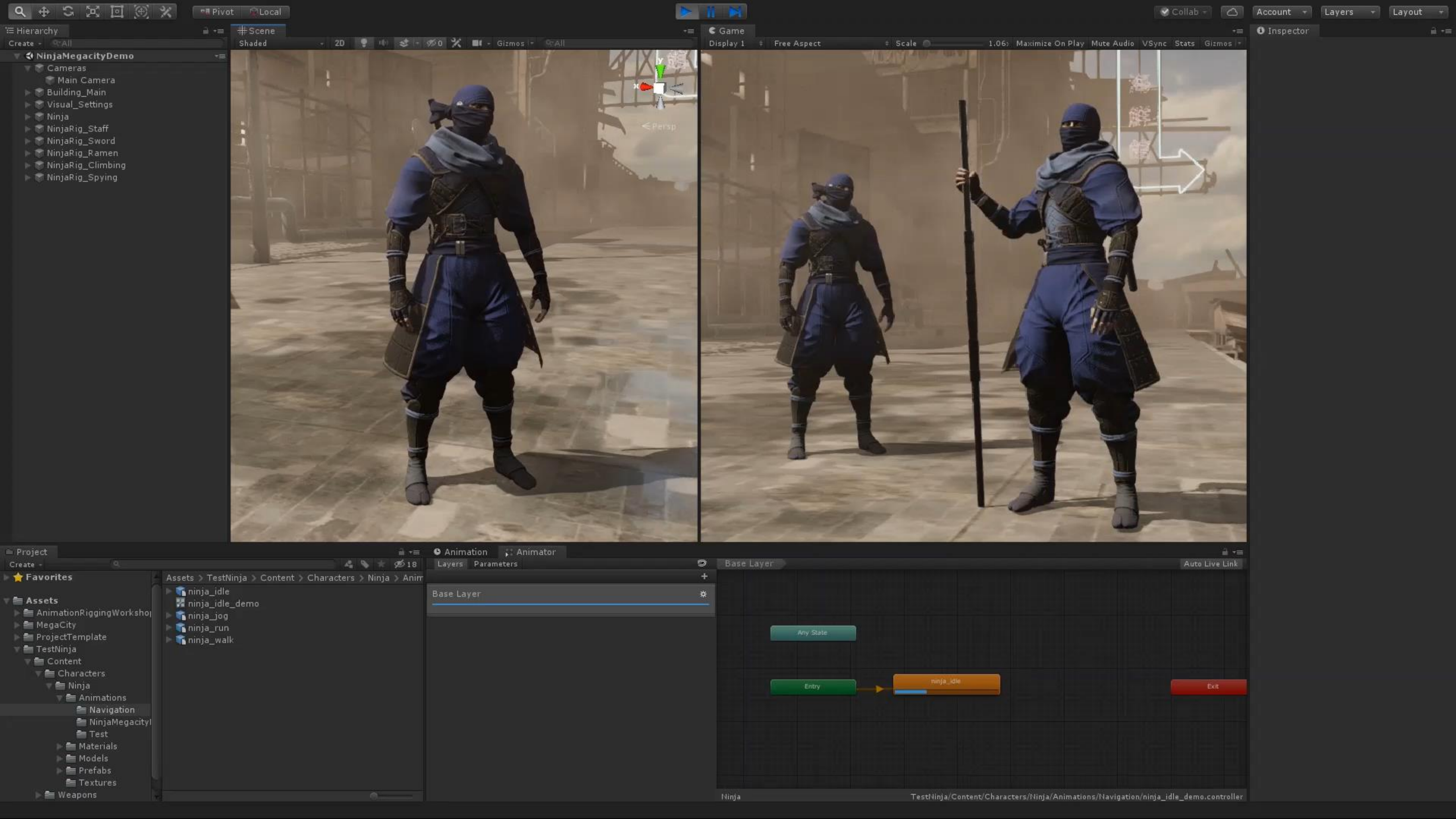
thrive SIGGRAPH2019 LOS ANGELES • 28 JULY – 1 AUGUST

NINJA MEGACITY SCENE

- Open the scene: NinjaMegacityDemo
- One animation clip → many rig variations
  - Animated rig overrides
  - Interact with the rig in Play mode 
  - Set keyframes in Animation Window Preview mode







Pivot
Local
Collab
Account
Layers
Layout
Hierarchy
Create
NinjaMegacityDemo
Cameras
Main Camera
Building_Main
Visual_Settings
Ninja
NinjaRig_Staff
NinjaRig_Sword
NinjaRig_Ramen
NinjaRig_Climbing
NinjaRig_Spying
Scene
Shaded
2D
Gizmos
Persp
Game
Display 1
Free Aspect
Scale
1.06
Maximize On Play
Mute Audio
VSync
Stats
Gizmos
Inspector
Project
Create
Favorites
Assets
AnimationRiggingWorkshop
MegaCity
ProjectTemplate
TestNinja
Content
Characters
Ninja
Animations
Navigation
NinjaMegacity
Test
Materials
Models
Prefabs
Textures
Weapons
Assets > TestNinja > Content > Characters > Ninja > Anim
ninja_idle
ninja_idle_demo
ninja_jog
ninja_run
ninja_walk
Animation
Animator
Layers
Parameters
Base Layer
Auto Live Link
Any State
Entry
ninja_idle
Exit
Ninja
TestNinja/Content/Characters/Ninja/Animations/Navigation/ninja_idle_demo.controller

Unity Exhibitor Sessions, Room 407

- Introduction to Animation Rigging for 2019.2 @ 3:30pm
- Extending the Animation Rigging Package with C# @ 4:00pm

THANKS!

We look forward to hearing your feedback on Animation Rigging

https://forum.unity.com/forums/animation-previews.141/

Dave Hunt, Technical Artist, davehunt@unity3d.com

Simon Bouvier-Zappa, Animation Developer, simonbz@unity3d.com

Olivier Dionne, Animation Developer, olivierd@unity3d.com