





Miarmy Manual

Version 1.5

Basefount Software

BASEFOUNT TECHNOLOGY (HONG KONG) LIMITED

©2011-2012 Basefount

Table of Contents

Brief	12
Structure.....	12
Part 1 Main Concept.....	13
Miarmy Human Logic Engine Concept	13
Decision Node.....	13
Defuzz Process.....	14
Solving Decisions Active	14
Make Priority for Decision Nodes.....	15
Agent Brain Instantiate.....	16
Assets Management	16
Miarmy Contents.....	16
Agent Groups.....	17
Perception_Set Group	19
Placement_Set Group.....	19
Naming Conventions	19
Animation Rig	19
Original Agent, Action, Action Shell, State and Decision.....	19
Bounding Box and Action Proxy	20
Geometry.....	20
General Pipeline	20
Plan.....	20
Prepare	21
Create Agent Infrastructure	21
For Logic and behavior	23
Rendering and output	23
Solver.....	24
Solver Contents	24
Simulation Pipeline.....	24
Reset, Update, Pause and Break Mechanism.....	25
Non-Connection Sequential Node Communication	25
Part 2: Menu Items.....	26

Part 3: Agent Infrastructure	53
Rig	53
Bone structure	53
About binding	55
Original Agent.....	55
What is Original Agent? And why we need it?	55
Creating Original Agent	56
Agent Group Node and Agent Type	56
Render Geometry Binding	58
Render Geometry Management	58
Render Geometry Shader.....	58
Bone Shape for Display & Physical Simulation	59
Bone Attribute & Flags	60
Physical Joint Type.....	60
Physical Joint Direction.....	61
Physical Joint Limits	61
Placement.....	64
Point & Formation Placement	65
For “formation” placement:	68
Populate Agents from Pace Node	70
Curve placement.....	70
Terrain Attachment + Point placement.....	71
Proportion in Place Node	72
Hierarchical placement.....	73
Solver Space.....	74
Inverse Placement	74
Place Node Data Structure	76
Animation and Action.....	78
Animation to Action Pipeline Suggestion	79
Create Action Node	79
Transition Map Building	82
Concept.....	82

Main Features of Transition Map	84
How to Build Transition Map.....	86
Decision	90
Create Decision node	90
Agent	90
Agent attributes.....	91
Agent Mode	93
Agent Memory Data	93
Part 4 Logic in-depth	94
Fuzzy Logic.....	94
Pipeline	96
Terminology.....	96
General Pipeline	96
Engine Process Breakdown.....	97
Implementation	99
Step 1: Get Channel Results	99
Step 2: Fuzzy Logic for a Single Sentence Calculation	100
Step 3: Fuzzy Logic between Sentences Calculation	102
Step 4: Fuzzy Logic for Priority Ranking and Interfere to Node Active	104
Step 5 & 6: Output Mechanism - Node Active, Default Decision and Normal Decision	106
Part 5 Logic & Perception	109
Logic Presets.....	109
Channel Presets	109
Decision Presets.....	110
Interactive with Agent.....	110
Sound.....	110
Vision	114
Agent Exclusion Feature	117
Dynamically Decrease Intelligence Mechanism (DDIM 1.0).....	118
Collision Detection.....	118
Interactive with Environment.....	119
Ground.....	119

Bound	121
Indexing Technique	122
Zone	123
Spot.....	127
Road.....	128
Maya Field	137
Composition of Forces.....	138
Maya Fluid	139
HP & MP	139
Scene Info	140
Randomize	140
Simulation Space	142
World Transformation for physical simulation	142
Local Transformation for non-physical simulation.....	142
Solver Space Node.....	143
Output Behavior Channels	144
Transform Speed	144
Bone Offset.....	145
Sound.....	145
Vision	145
Aim Constrain	146
Action.....	150
Ragdoll	150
Watch Logic Results.....	151
Brain Viewer	151
Feedback in Essential.....	152
Part 6 Action	153
Action Infrastructure	153
Action Node Data	153
Action Node Attributes for Playback and Transition.....	154
Action Channels.....	159
Action Trigger	159

Action Modifier.....	160
Action Drive Agent.....	161
Pipeline	161
Getting the Target Action	161
Parsing out the Next Action.....	162
Action Transition	164
Action Transition Visualization	164
Action Proxy List	166
Part 7 Physics.....	168
Enable Dynamics.....	168
Dynamics Types	168
Dynamics Channels.....	169
Collision Detection.....	171
Collide Check Channels.....	171
Pre-build RBD Objects	176
Pre-build RBD Objects for Collision Detection	176
Pre-build RBD Objects for Dynamical + Action Hybrid Drive.....	177
Pre-build RBD Objects for Cloth Collide	178
Optimization	178
Dynamics Features	178
Physics Global	178
Step Time and Sub Step.....	179
Gravity VS Mass	179
Terrains and Default Terrain	180
Maya Field	183
Force Field General Rules	184
Maya Fluid	184
Build-in Force Field	185
Combo	187
Kinematic Primitives	188
Cloth simulation.....	189
Mute Dynamics.....	193

Part 8 Render.....	194
Agent Cache.....	194
Creating Agent Cache	194
Agent Cache for Bone Rotate Only.....	195
Agent Cache with translate	195
Agent Cache with cloth.....	195
Agent Cache Format	196
Mesh Drive	196
Concept.....	196
Features:.....	197
Optimizing Mesh Drive before Implementation	197
Generate (Duplicate) Mesh	197
Pairing.....	199
Enable Mesh Drive.....	200
Render Solutions	201
Render by Renderman.....	201
3Delight Configuration	201
Manually Setup Renderer (Windows)	202
Pipeline	202
How to Render.....	203
Render Agents	203
Random Geometries Rule.....	204
Texture Naming Convention and Random Textures	206
Render Global.....	208
Texture generate and path accessible.....	208
Multi cam render.....	209
Preset shaders	209
Self-defined Renderman Shader	215
Light Usage & Shadow	217
Motion Blur.....	217
Split render	219
Procedural Primitives	221

Subdivision Mesh.....	224
Matte	225
Fetch Agent Render Information from Miarmy to Your Own Pipeline	225
Part 9 Optimization	227
Low Efficiency Channels Optimization	227
Vision	227
Collide Family Channel & Pre-build RBD Object.....	229
Collide Any Channel.....	230
Bounding Box Display	230
Solver Space.....	231
Easy Transition.....	231
Optimizing Crowd Dynamics	231
Using Spot or Zone Trigger Dynamic	231
Mute Dynamics.....	232
Part 10 Expanding Guide	234
Scripts	234
Build your own logic Preset	234
Build your own channel Preset.....	234
Build your own automatic original agent setup pipeline	234
Build your own automatic action creation pipeline	234
Particle Express for Hybrid Dynamics Effects	234
API.....	234
Write your own Field Node	234
Render Fur	234
Use Miarmy Engine Drive Maya Character	234
Miarmy Reference	235
Callback Functions	235
After Load Miarmy.....	235
Before Save.....	235
After Save	235
Selection Change	235
Before Duplicate	235

Cloth 236

Channel Reference List 237

Brief

Thank you for reading this. This document listed and explained every detail of Miarmy. Before you reading this, please make sure you already know some basic of Miarmy. This document explained many things in technical under the hood and not a step by step tutorial. If you like to know or learn Miarmy from start, please refer the PDF *Miarmy Example tutorials* and Miarmy Online video tutorials www.youtube.com/basefount.

Structure

We organize this document like this:

- **Part 1 Main concept:** This part explained the logic engine concept of Miarmy, the Miarmy infrastructure contents and general workflow and pipeline for making project
- **Part 2 Menu Item:** This part detailed listed all the menu item of Miarmy
- **Part 3 Agent Infrastructure:** This part listed and explained the agent contents we need create and how to make them work.
- **Part 4 Logic in-depth:** This part explained the logic workflow and pipeline in-depth and listed all the logic algorithms of Miarmy Human Logic Engine.
- **Part 5 Perception & Logic:** This part explained all the perception contents under the hood workflow and algorithm, such like road, zone, etc.
- **Part 6 Action:** This part explained the action node, action transition mechanism and drive action in logic.
- **Part 7 Physical Simulation:** This part explained the physical collision detection and dynamic simulation and some more stuff under the hood.
- **Part 8 Rendering:** This part explained the Miarmy caching mechanism and rendering workflow
- **Part 9 Optimizing:** This part explained in-depth how you can optimize your scene.
- **Part 10 Expanding:** This part explained using Script and C++ API for expanding Miarmy. (coming soon)

Part 1 Main Concept

In Miarmy, we get rid of any node connections and programming, balanced the simplicity and diversity, and dedicated designed a fuzzy logical based human logic engine for imitating how people thinking. And you can easily construct complex logic just by several nodes.

Miarmy Human Logic Engine Concept

Basically, people think things just like this, “If there is something happening, decide to do something”. For instance, there are some obstacles in front of me, I decide to turn away and go around them. People always make some decisions based on what is happening.

More specific, each “decision” contains “something happening” and “the next things want to do”. In Miarmy, we call them “conditions (sentences input)” and “decisions (decisions output)”. People always make some decisions based on the conditions.

There may be many things happening together, so, we maybe have many decisions. From these decisions we need figure out which is the correct next things to do. The process from decisions to behaviors is called “defuzz” in Miarmy engine and this process is total automatic.

Additionally, in the process of defuzz, there may be several decisions conflicted. So, people will naturally make priority for them. For example, someone in my left and near, I want to go right, but there is a cliff on my right, without saying, I have to go to left for avoiding the fall. In this situation, go left and go right are conflicted decisions, we need make a priority for them. Avoid cliff take higher priority than avoid each other. People always make priority based on natural principles, and finally decide what to do next.

Generally, Miarmy Engine was built for trying to simulate the above brain process in human brain, conditions, decisions, priority ranking and behaviors.

In Miarmy,

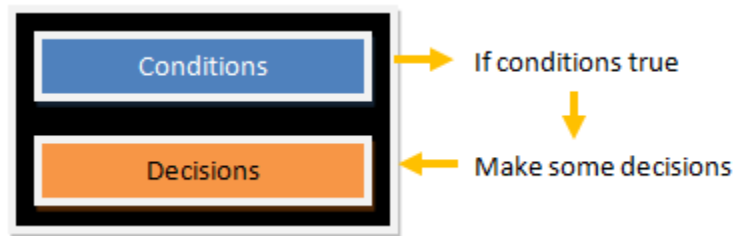
- We use “decision node” to imitate the “the conditions and decisions”,
- We use parenting hierarchy to rank priority for multiple “decision nodes”.
- We make the “defuzz” (from decisions to behaviors) process totally automatic.

Decision Node

Decision nodes are the basic units for simulating the Human Logic. In each one of decision node, there are

- Several conditions tests
- Some decisions to make.

We usually call these conditions as “sentences” and call the decision results as “outputs”. Miarmy engine will test the “sentences” true or false with fuzzy logic, and then figure out whether and how much decides the “outputs”.



Decision Node Pipeline

Defuzz Process

Defuzz is the process based on your decisions figure out the behaviors, and this process is totally automatic. In this process we need each **decisions active** and **decisions priority**. After that, we can assign the behavior to the digital actor.

Solving Decisions Active

The decision active is the condition test result of decision node. For example, if someone in my left and near, I need go right and slow down.

- Input Sentences:
 1. Someone in my left
 2. Someone is near
- Output Decisions:
 1. Decide go right
 2. Decide slow down a bit

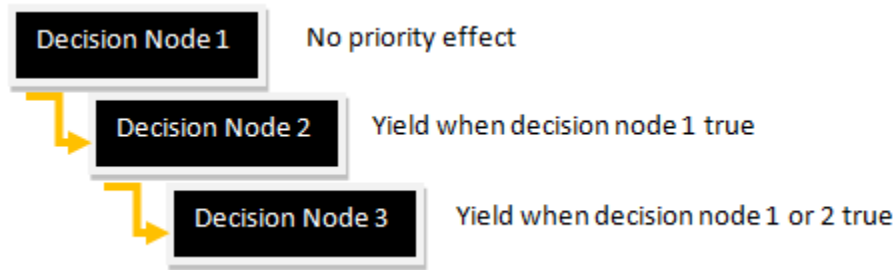
In this example, the decision active of this decision nodes is the condition test result of these input sentences. We use the fuzzy logic to test conditions, and the result value will be a float value from 0 to 1. We will talk about the sentences test later in detail. See **Part 4 – Logic in-depth**

Active	Priority	Logic	Not	ID	Input	Inf	Min	Inf	Max
<input checked="" type="checkbox"/>	0	&&	<input type="checkbox"/>	A	sound.x	<input type="checkbox"/>	-90.0000	<input type="checkbox"/>	0.0000
<input checked="" type="checkbox"/>	0	xor	<input type="checkbox"/>	B	sound.d	<input type="checkbox"/>	0.0000	<input type="checkbox"/>	0.0000
<input type="checkbox"/>	0		<input type="checkbox"/>			<input type="checkbox"/>	0	<input type="checkbox"/>	0
Parse Result: (A ^ B)									
Active	Decision	Value							
<input checked="" type="checkbox"/>	ry	100.0000		Auto Fill					
<input checked="" type="checkbox"/>	walk:rate	0.5000		Auto Fill					

Blue bound for conditions and yellow bound for decisions

Make Priority for Decision Nodes

For achieving the priority for decisions, we just need simply re-arrange the hierarchy of decision nodes.

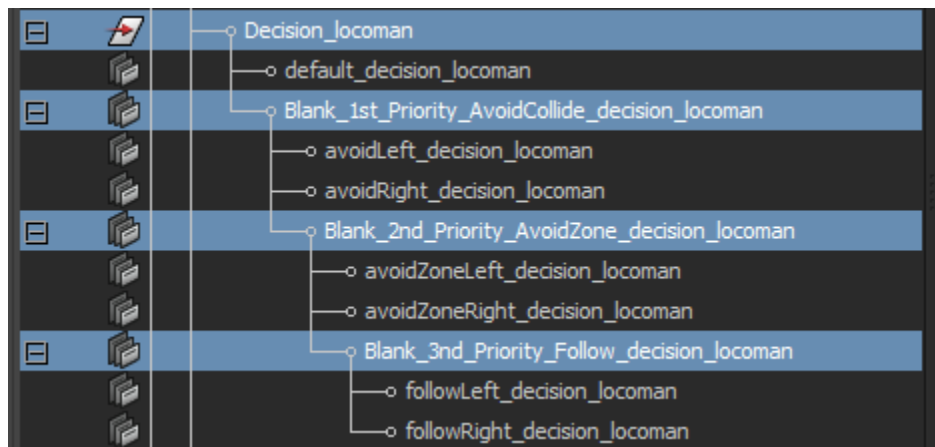


Decision node priority ranking pipeline

In the following real example, character need to avoid each other, get around a zone and follow the road direction. We need make the following priority ranking:

1. First priority: avoid each other
2. Second Priority: get around a zone
3. Third Priority: follow road

Just re-arrange the hierarchy of these nodes we can achieve this:



Priority ranking by hierarchy

You can also group a bunch of decision nodes together and make this bunch of decision nodes totally lower priority than the other bunch of nodes.

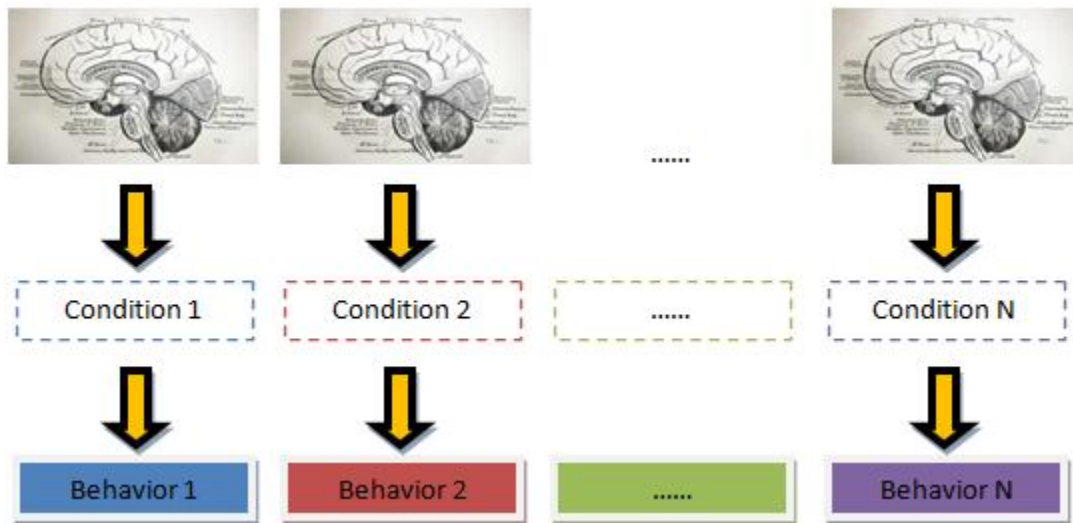
All of decision making process and priority ranking process are driven by Fuzzy Logic engine. We will explain the in-depth algorithm and implementation in Part 4: Logic in-depth

Agent Brain Instantiate

You may ask, a group of agent, all of them have the same brain, are their behaviors also the same??

No. The behavior is depending on the decisions, and the decisions are depending on not only the brain, but the environment. Different agents are facing different environments. Therefore, they will have different decisions and behaviors

Just like the picture below.

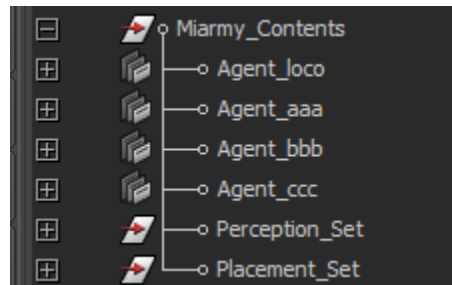


The same brain in different condition, brings out different behaviors

Assets Management

Miarmy Contents

Miarmy_Contents Group is the main group contains all of Miarmy infrastructure. They are agent groups, Perception_Set group and Placement_Set group



Miarmy Contents, General Group

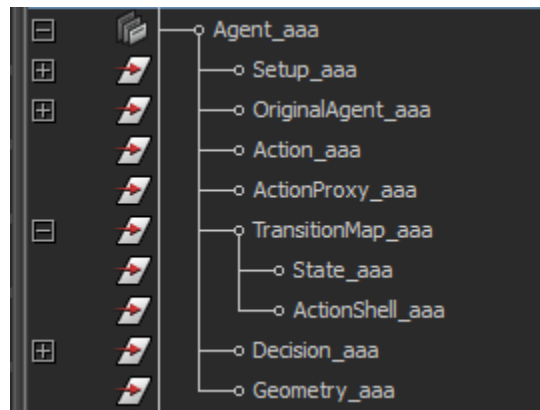
Agent Groups

Usually there may not be only one kind of agent in scene. We group their contents in different groups and this kind of group is McdAgentGroup node, we dedicated designed the McdAgentGroup rather than using group node (Maya transform node), because we want them can be searched by engine easily. Technically, McdAgentGroup node is the same as transform node. In addition, there are many extra attributes on this node. And we call them “Agent Type Attributes”



Extra attributes on McdAgentGroup node

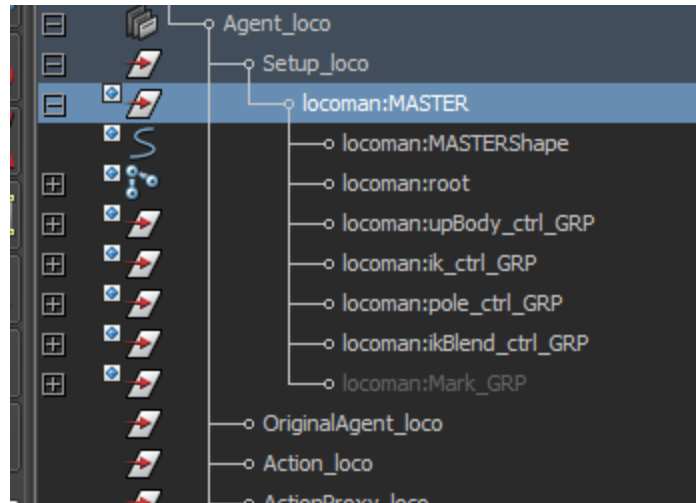
And for each type of agent, there are several kinds of contents in different groups. The contents in each agent group node, only belongs to their specific agent type. They are shown in following image, and we will introduce them in details.



Agent Group Structure

Setup_<agentName> under Agent Group Node

This group contains the user defined character rig. And this rig is usually should be referenced into Maya scene for keeping you scene clean, because this rig will not be useful when all agent infrastructure created. We just need reference its bone structure for easily deleting after everything is done.



A rig been referenced to scene under Setup Group

Original_<agentName> Group under Agent Group Node

This group contains a bone structure with exact the same as the character rig. Original agent is the template of agent and contains many necessary attributes which character rig doesn't have. When we place the agent out, we also need the original agent to provide some extra information, such like the render, cloth, matching information. We will continually provide more information throughout this entire document.

Action_<agentName> under Agent Group Node

This group contains all the action nodes, the action nodes are generated from animated Character Rig and responsible for drive the agent to the target poses. For more details about the action nodes, please check out the "action" chapter.

ActionProxy_<agentName> under Agent Group Node

This group will contain an only one node if you create action proxy. The action proxy is a fairly simple node only contains a list of action names. If you already create some actions for this type of agent, you can fill some existed action names into action proxy list and our engine will take place the logic procedure and perform the actions inside this list cyclically, one by one. For more details about action proxy node, please check out the "action Proxy" chapter

TransitionMap_<agentName> under Agent Group Node

This group contains the action shell and state nodes. They are the element nodes of the Transition Map. For more detail about the action transition, please check out the "Transition Action" and "Transition Map Viewer" chapter.

Decision_<agentName> under Agent Group Node

This group contains all of the logical decision for this type of agent. With these nodes, we define the agents' brain and behaviors

Geometry_<agentName>under Agent Group Node

This group contains all the geometries of this agent type need to be rendered. Just using the simple structure arrangement you can achieve the random geometries. For details, please check out the “random geometry” in “render” chapter.

Perception_Set Group

All of the perception contents node will be located in this group after them firstly been created, there are lot of types of perception content, such as road, bound, zone, field, etc. For more detail, please check out the “Perception Contents” chapter

Placement_Set Group

All of the place nodes will be located in this group after them firstly been created. Place node is a pre-visualization of the agents would to be placed. This node will provide a visible position and rotation for the agents to be populated and let the users setup their transformation and proportion. For more details, please check out the “Placement” chapter

Naming Conventions

Note: This naming convention is only available in current version. In future, the naming might be more arbitrary.

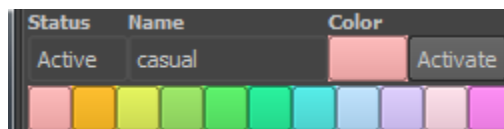
Animation Rig

Rig joints and controls can be any naming including namespace

Original Agent, Action, Action Shell, State and Decision

<Name>_<Node Type>_<Agent Type Name>

Example:



The agent type name is casual

Original Agent Joint: torso1_ogb_casual

Action Example: standToRun_action_casual

Decision Example: default_decision_casual

State Example: walk_state_casual

Action Shell Example: jog_actionShell_casual

Bounding Box and Action Proxy

There is unique bounding box in each type of agent, and also Action Proxy node

<Node Type>_<Agent Type Name>

Example:

Original Agent Bound Box Example: BoundingBox_casual

Action Proxy Example: actionProxy_casual

Geometry

Geometry cannot contain namespace comma “:”, except, it can be any name

General Pipeline

In this chapter we will introduce the general pipeline of all Miarmy throughout. But this is just a general introduction, if you want to know the details, please refer the specific part and chapter.

Plan

Crowd simulation sometimes is a little bit complex that you have to consider the entire pipeline throughout from modeling to final rendering. We need plan it carefully before we start it. Here we listed some important stuffs you need plan them in details before starting.

Models complexity for rendering speed

If you have many complex models for each agent, it is going to be difficult for rendering. So please plan the complexity of your models based on the shot and agent number.

How to do the animation and Action

You need consider which action you need the agent have, so you need prepare them before started, you can use any type of animation keyframe, preset packages or motion capture are ok.

Logic complexity

We need carefully plan which the behaviors we want to the agents perform and which the logic can achieve these. Then we can plan which decision the agents need make. And which the perception we have to create. If the scene is too large, you need consider using the Decrease Intelligence Mechanism or making some agents with very low-intelligence for filling the scene.

Agent types

You need consider how many types of agent you need, such as soldier and enemy. Each one has a color and some specific attribute such like sound range, vision range, density etc.

Whether we need perform crowd dynamics

If you want to let your agent been enable ragdoll dynamics, you need plan them from start. When agents perform dynamics, each of the character bone will be transformed to RBD, so each bone should have length. Mute some agents are never going to be enabled dynamics for saving huge time. Plan the collide body etc.

Which renderer you need to use

For current version, if you want to render you scene by Renderman, you need install 3delight.

If you want to render you scene with Mesh Drive and arbitrary type of renderer, you need first consider the complexity of your character model because mesh drive need duplicate all the geometries out and drive them. See “Mesh Drive”

Prepare

Rig

In this step we usually reference rig to scene, or if you want, you can build your own one. The rig has to obey some limits, please check out the “rig” chapter

Animation

We create action node for agent from animated character rig with same structure, so you need prepare these animated character before starting.

Create Agent Infrastructure

Original Agent Create and Check

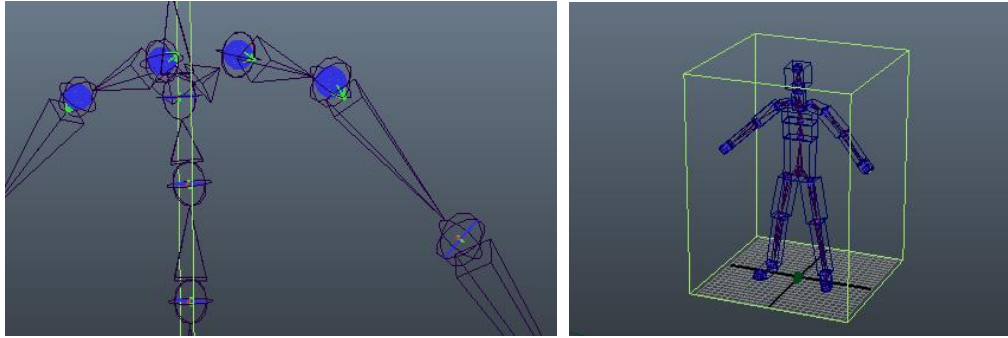
In this step we need build original agent from character rig. If everything of rig in all right, the original agent will be the exact same bone structure as character rig, without all of the controls like the IK solver and constrains. The original agent is the blueprint template of agent. For more about the original agent, please check out the “original agent” chapter

Original Agent Refine

Then we need fine-tune the original agent in many aspects. Setup bone attribute, setup agent attribute, adjust the bounding box, set flags, etc.

Physical Setup

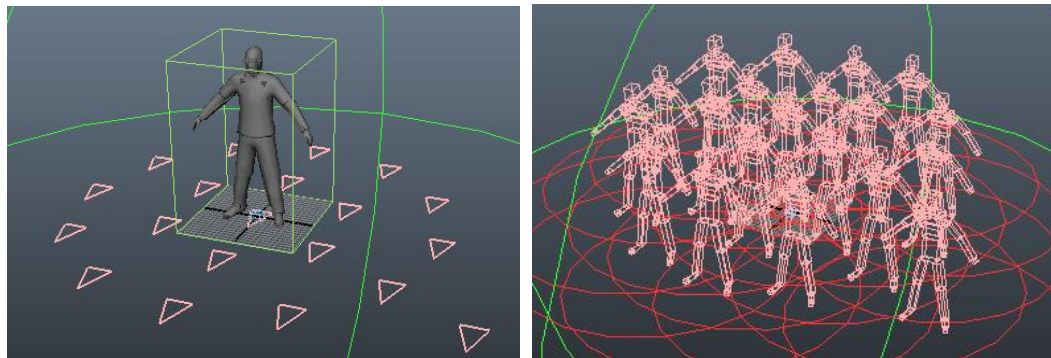
If you want to your agent can be enable dynamical ragdoll or be influenced by the force field, you need setup the physical bones and joint correctly and carefully. Also you need setup the collision detection flags (see “Original Agent” and “physics simulation” part)



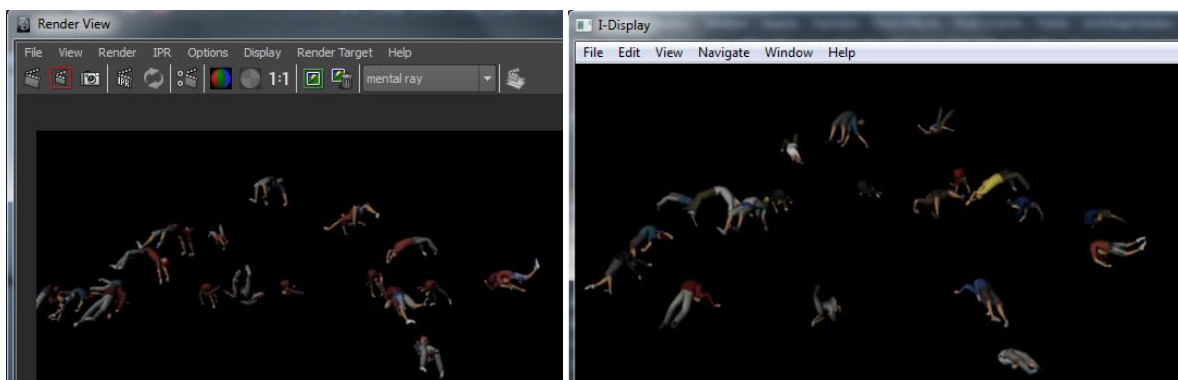
The original agent, left: physical joints, right: original agent bounding box

Simple Render Test

You need skinning all of the geometries to original agent so that our engine can render it. In this step, we usually create a very simple placement node and place the agent out and render some agents.



A simple setup and placement



Test render by mental ray and 3delight

For Logic and behavior

Create Perception & Logic

In this step we need create the planed logic for the agent, and make perception contents like road, zone into your scene.

Test simulation, fine-tuning, and optimizing

In this step we need place part of the agents out and check is that everything running ok. Please notice, we are not recommended you place full number of agents for now. Small group of agents are easier for debugging and designing the logic of them.

Usually you need find-tune the data in logic node for achieving better results. Also, you can continually add perception contents and logic in any time even in simulation playing back.

Optimization

You can optimize your scene in many aspects. You can decrease the intelligence of many agents in real-time, mute some agents' dynamic etc. See dedicated Part 9, Optimization

Generate Crowd and Preview

After you satisfying your setup, you can place all of your agents out and check out the entire scene in your shot

Rendering and output

Caching

After caching, you can achieve interactive playback for your large scene. You can easily watch your finally results. Also, the deformation motion blur can be rendered out only after caching.

Get rid of error agents

If some of agents are not correct, we recommend you get rid of them out of the shot. But you cannot delete the agent directly because this agent will appear again after next placing. What you need to do is using a Maya expression to put these agents out of the screen. Just like "McdAgent1.ty = 100000;"

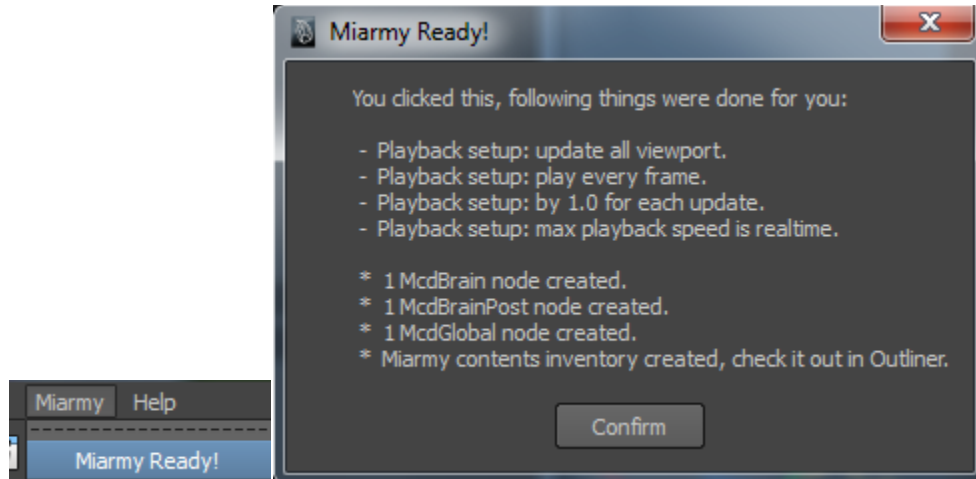
Render it!

If you satisfy your scene, you can render it by 3delight directly or duplicate these geometries out and render them by any other renderer. You also can make geometry cache or Alembic cache for exporting to others package for VFX and rendering.

Solver

Solver Contents

When you click the Miarmy > Miarmy Ready, the solver contents have been created, they are “McdBrain”, “McdBrainPost” and “McdGlobal”, without these nodes, your simulation is not going to start.

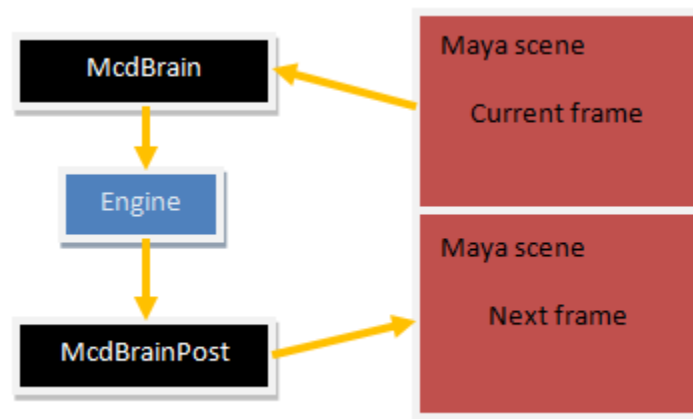


Solver setup and contents creation

Simulation Pipeline

Each time update, Miarmy will do the following things if the simulation is not broken.

1. The McdBrain is responsible for collecting the information from scene. Such as the agent information, the perception contents information, etc.
2. Then, McdBrain will send the collected data to engine.
3. The engine will calculate out the result, such as the agent transformation and posing.
4. The McdBrainPost is responsible for updating the scene using the engine results



Simulation Pipeline

Reset, Update, Pause and Break Mechanism

There are 4 statuses of solver which are “reset”, “update”, “pause” and “break”.

Suppose that the start time of engine is 1, we can set it by `McdBrain.startTime`



Green: reset, Yellow: update, Red: will break simulation

Take a look at the green point, any time you can reset your scene by setting current time to start time (or before start time)

As the yellow arrows shown, the engine can simulate and update Maya scene correctly, when playback from start time and frame by frame.

If you jump frame back or jump more than one frame, the simulation will break, just like the red arrows shown. At this time, the engine will never solve until you reset it (back to start time).

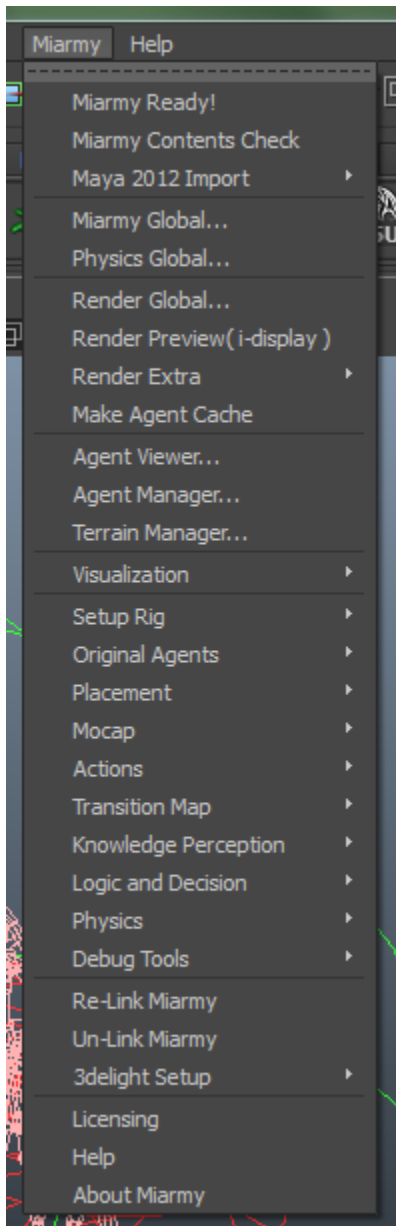
If simulation has been paused at the “x”th frame, you can jump time as you like, but before resuming it, you need set the time when the time simulation break which is the “x”th frame. You can pause the simulation in Miarmy Global.

- **Reset:** the agents will be set back to initial status, meanwhile, some information in agent memory will be clear and reset, such as the physical stuff, cloth buffer, etc.
- **Update:** simulation in progress, engine is still working for updating scene
- **Pause:** simulation stop but can resume later
- **Break:** simulation will not continue proceed any more until reset.

Non-Connection Sequential Node Communication

Using some non-traditional Maya techniques, we designed a non-connection sequential node communication solver. So, there are no connections among any Miarmy nodes for data transferring. All of data transfer is happening in memory. When we create the Miarmy scene, we don’t need hypergraph anymore.

Part 2: Menu Items



Miarmy Ready!

Click this button system will do the follow things for you:

1. Setup Playback Options:

- a. Update all viewport
- b. Play every frame
- c. By 1.0 frame for each update
- d. Max playback speed is realtime (depending on the user specify, 24fps or 30fps)

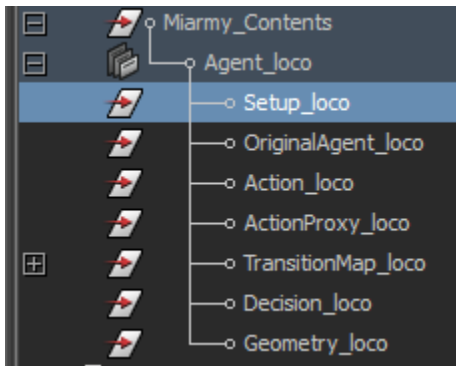
2. Create Something:

- a. Create a McdBrain node if not exist

- b. Create a McdBrainPost node if not exist
- c. Create a McdGlobal node if not exist.
- d. Create Miarmy_Contents group and some preset stuffs
 - **Miarmy Contents:**
 - **Agent group 1:** agent repository
 - **Setup:** rig location (usually reference into the scene)
 - **Original Agent:** template for current agent
 - **Action:** Miarmy action generated by Maya animation
 - **Action Proxy:** an action list node can proxy all action playback
 - **Transition Map:** state and action for build transition map
 - **Decision:** logic entity
 - **Geometry:** binding geometry
 - **Agent Group 2**
 - ...
 - ...
 - **Agent Group 3**

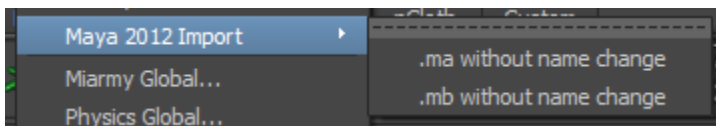
Miarmy Contents Check

It will check the naming convention and integrity of Miarmy_Contents group and stuff inside.



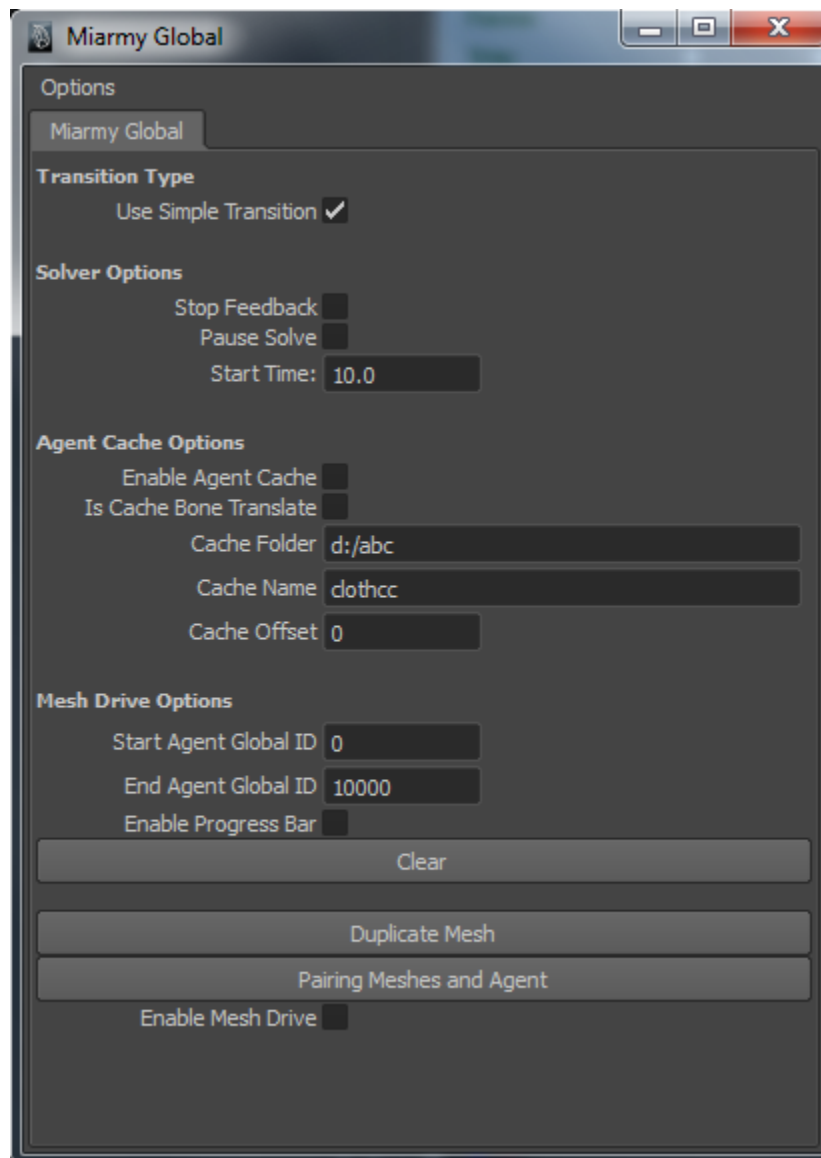
It will try to fix the naming in specific group.

Maya 2012 Import



Maya 2012 cannot import contents without name prefix or namespace. This tool let you import contents without changing the name of objects.

Miarmy Global



- **Transition Type**

- **Use Simple Transition**

If enable, the transition map will be ignored by engine. Transition will automatically happened by themselves between any 2 actions.

- **Solver Options**

- **Stop Feedback**

If enable, the agent will not send information to Brain Viewer. It will save your time if you are already done the logic construction.

- **Pause Solve**

If enable, the simulation will be paused. And if you disable this at the frame you enable this, the simulation will continue.

- **Start Time**

Specify the simulation start frame.

- **Agent Cache Options**

- **Enable Agent Cache**

Enable agent cache, make the engine find cache file instead of solving

- **Is Cache Bone Translate**

If enable, when making cache, the translate information will be recorded. When assigning, the translate information will be set to bone. It's useful for record break body like the sword and shield

- **Cache folder**

Specify in which the cache file located, the path should be accessible

- **Cache Name**

Specify the cache file name such as "scene001cache"

- **Cache Offset**

Frame offset when apply cache to scene.

- **Mesh Drive Options**

- **Start Agent Global ID**

Specify the first agent which will perform mesh drive

- **End Agent Global ID**

Specify the last agent which will perform mesh drive, mesh drive will enable the agent id between these 2 values

- **Enable Progress Bar**

Show updating progress bar when agents drive their meshes

- **Clear**

Delete all duplicated meshes

- **Duplicate Mesh**

Duplicate meshes for agents, from start ID to end ID

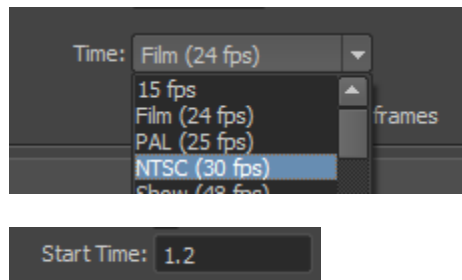
- **Pairing Meshes and Agent**

Mapping agent to their paired geometries in agent memory

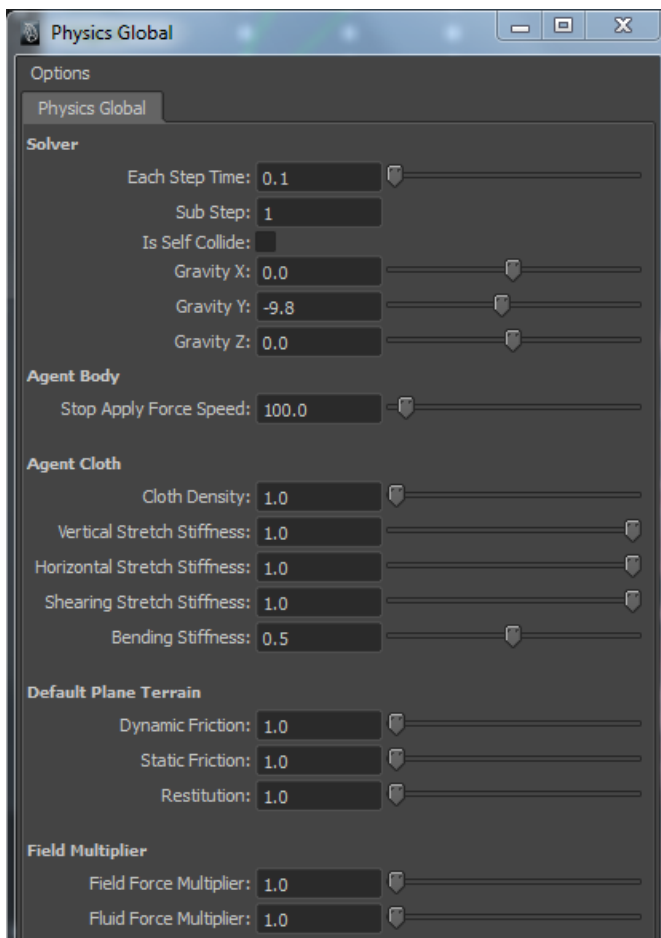
- **Enable Mesh Drive**

Enable mesh drive for update meshes.

Note: if you change your frame rate of this scene in Maya Preferences, the start time will change. You need to fix it to the correct value.



Physics Global



- **Solver**
 - **Each Step Time:**
Specify how much time will proceed in one Maya frame
 - **Sub Step:**
Specify how many steps will make in Each Step Time
 - **Is Self-Collide:**
Specify whether you need enable collision on bones of single agent
 - **Gravity X:**
Gravity acceleration in X direction
 - **Gravity Y:**
Gravity acceleration in Y direction
 - **Gravity Z:**
Gravity acceleration in Z direction
- **Agent Body**
 - **Stop Apply Force Speed**
When the speed of RBD exceeds this value, the force field cannot add force on to it.
- **Agent Cloth**
 - **Cloth Density**
The particles' density of cloth
 - **Vertical Stretch Stiffness**
Stiffness in vertical direction, bigger value leads harder deform in vertical space
 - **Horizontal Stretch Stiffness**
Stiffness in horizontal direction, bigger value leads harder deform in horizontal space
 - **Shearing Stretch Stiffness**
Stiffness in diagonal direction, bigger value leads harder deform in diagonal space
 - **Bending Stretch Stiffness**
Stiffness for bending, bigger value leads harder bending or curling.
- **Default Plane Terrain**

The attribute effect on the plane terrain default created.

 - **Dynamic Friction**
The friction force when object is moving on terrain
 - **Static Friction**
The friction force when object is before moving on terrain
 - **Restitution**
The energy of bounces back from collision, increase this make object bounce greater on the default terrain

- **Field Multiplier**

Exclusive adjust the force effect on the rigid body of Miarmy. If the Maya field or fluid affects both Miarmy contents and Maya contents, you can adjust these values to change the result only on the Miarmy contents.

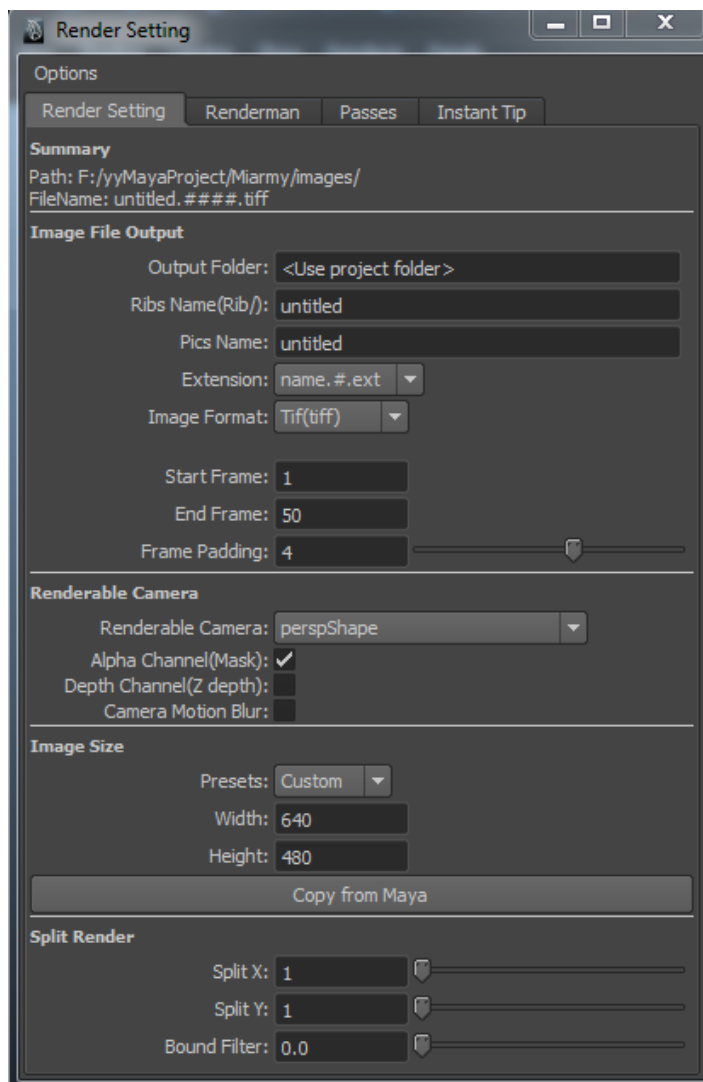
- **Field Force Multiplier**

The value will be multiply the result force effect on the rigid body from Maya field

- **Fluid Force Multiplier**

The value will be multiply the result force effect on the rigid body from Maya fluid

Render Global



- **Summary**

First line: The result folder name of output image or rib

Second line: The result file name of output image of rib

- **Image File Output**

The attributes can change the file export name, format and path

- **Output Folder**
The location your image files of each pass will output
- **Ribs Name(Rib/)**
The file name of RIB file
- **Pics Name**
The file name of image file
- **Extension**
The file name extension
- **Image Format**
The format of output image
- **Start Frame**
The frame the render will start
- **End Frame**
The frame the render will end
- **Frame Padding**
The bit of digits of file name endin

- **Renderable Camera**

Adjust the camera attributes

- **Renderable Camera**
Select the renderable camera here
- **Alpha Channel**
Enable this if you want to export alpha channel
- **Depth Channel**
Enable this if you want to export depth channel
- **Camera Motion Blur**
Enable this if you want ot apply motion blur when camera is moving

- **Image Size**

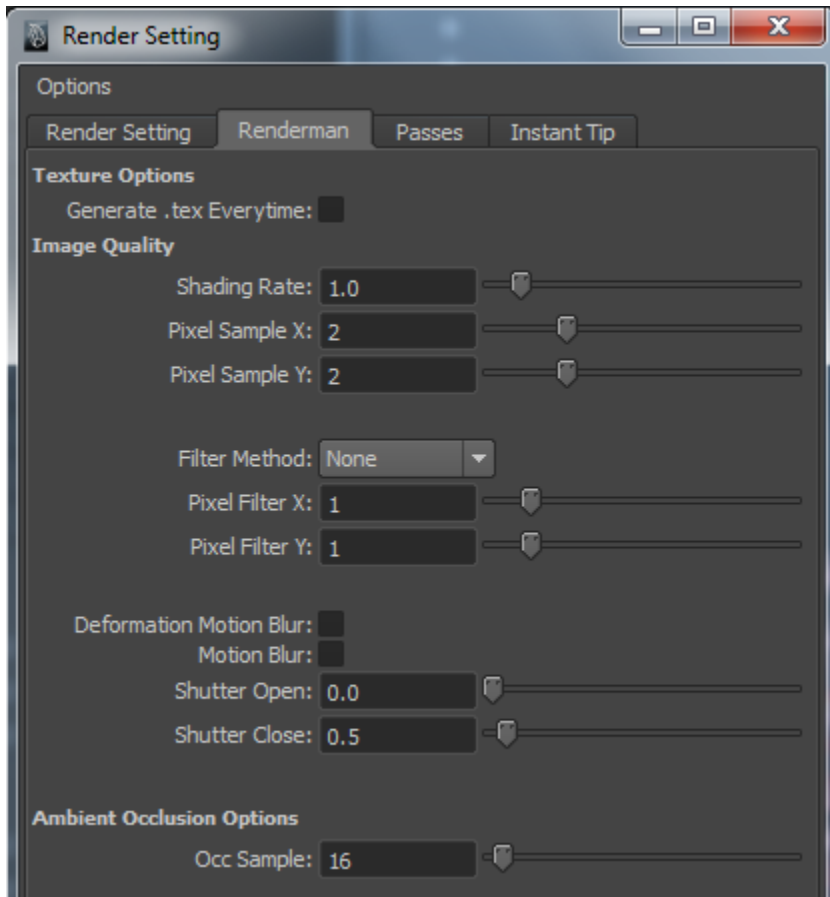
- **Preset**
Choose width and height presets
- **Width**
The X resolution of output image
- **Height**
The Y resolution of output image
- **Copy from Maya**
Copy the same setup from Maya Render Global node

- **Split Render**

To know more about split render, please refer the render part of this manual.

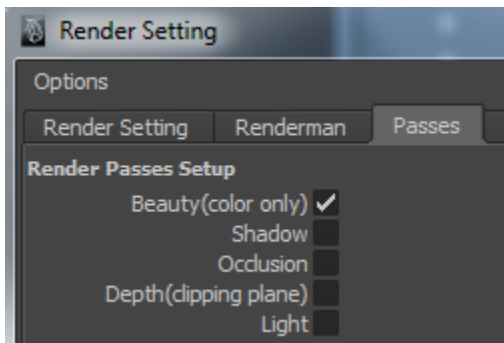
- **Split X**
The number of parts the viewport should be split in horizontal space

- **Split Y**
The number of parts the viewport should be split in vertical space
- **Bound Filter**
The filter pixels which between any 2 split parts



- **Texture Options**
 - **Generate .tex Everytime**
The system will generate a .tex texture file for each of texture file and send the .tex file to the RenderMan renderer when rendering. By default, the system will check the existence of that .tex file, if there is .tex file exist, system will not generate them again. If you need convert it every time, please enable this.
- **Image Quality**
The attributes exclusive belong to RenderMan and control the quality of output images
Please refer the RISpec if you want to know them in details
 - **Shading Rate**
Minimum rate of surface shading
 - **Pixel Sample X**
Sampling rate in the horizontal directions
 - **Pixel Sample Y**

- Sampling rate in the vertical directions
- **Filter method**
Antialiasing by filtering the geometry (or supersampling) and then sampling at pixel locations
- **Pixel Filter X**
The filter in horizontal directions
- **Pixel Filter Y**
The filter in vertical directions
- **Deformation Motion Blur**
Enable this to enable deformation motion blur, only available with agent cache
- **Motion Blur**
Enable this to enable motion blur
- **Shutter Open**
The times at which the shutter opens
- **Shutter Close**
The times at which the shutter closes
- **Ambient Occlusion Options**
 - **Occ Sample**
Increase this can get smoother result (less noise) but take more time to render

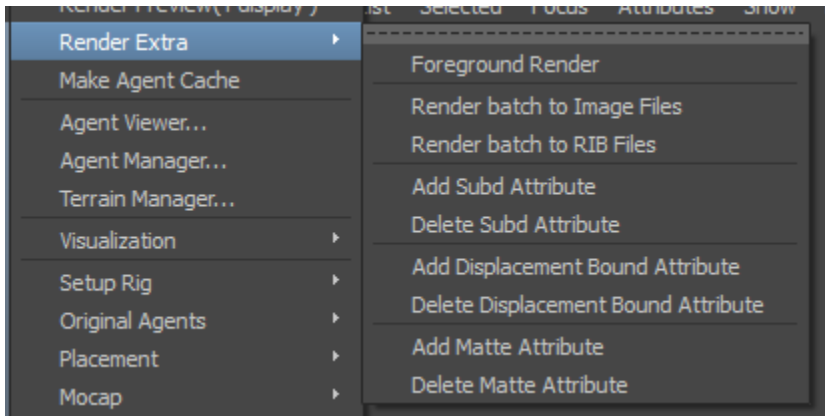


- **Render Passes Setup**
Specify which pass or passes should be rendered
 - If in preview mode (Miarmy > Render Preview), renderer will render the first option which enable here.
 - If in batch mode (Miarmy > Render Extra > Render Batch to Image Files/RIB Files), renderer will render all the options enabled here.

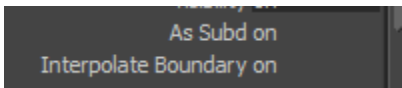
Render Preview (i-display)

Render current frame from current active viewport and put the image to the frame buffer (or i-display). The pass to be rendered should be the first pass enabled in Render Pass Setup of Render Global.

Render Extra (sub menu)



- **Foreground Render**
Open Maya Render View, and render this scene from start frame to end just inside of Maya
- **Render Batch to Image Files**
Render this scene from start frame to end frame and output the images to the folder been specified in Render Setting
- **Render Batch to RIB Files**
Convert the scene to the RIB files start frame to end frame and output the RIB files to the folder been specified in Render Setting
- **Add Subd Attribute**
Add 2 attributes to the selected objects, so that the objects can be recognized to the subdivision mesh by the render engine.

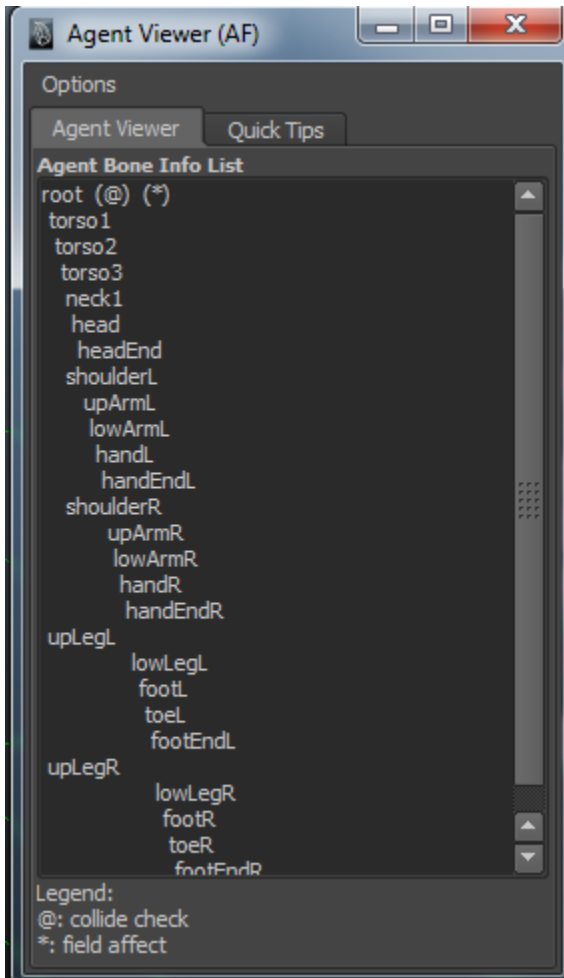


- **Delete Subd Attribute**
Remove the 2 attributes of subdivision mesh flags from selected objects
 - **Add Displacement Bound Attribute**
Add an attribute to the selected objects, so that the renderer will use this value when dealing with displacement shader
-
- **Delete Displacement Bound Attribute**
Remove the attribute of displacement bound from selected objects
 - **Add Matte Attribute**
Add an attribute to the selected objects, so that the renderer will render it totally black but alpha channel existed.
-
- **Delete Matte Attribute**
Remove the attribute of matte from selected objects

Note: if this attribute is added to the agent instead of geometry, the specific agent will matte out all the geometries it used

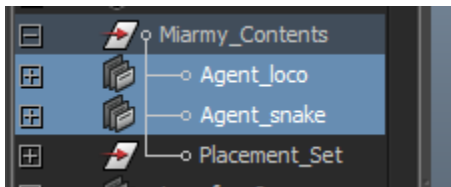
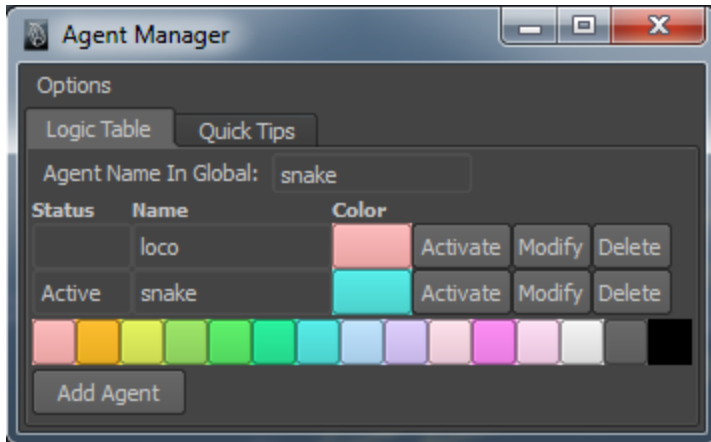
Agent Viewer (Auto Focusing)

The Agent Viewer will display the bone structure and some flags set in Agent memory



- **Bone structure:**
The name of bone in memory, it can be used in channel, like torso1:tx
- **Flags**
 - **(@) sign**
This bone will join collision check if using "collide" channel
 - **(*) sign**
This bone can feel field and fluid when dynamics is turned on

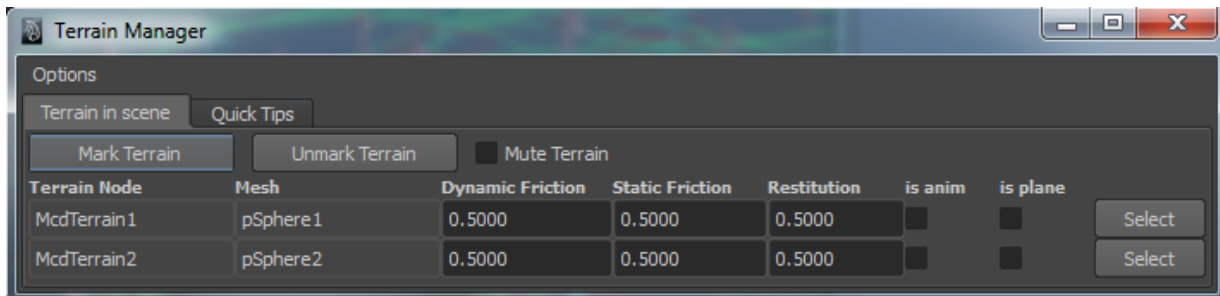
Agent Manager



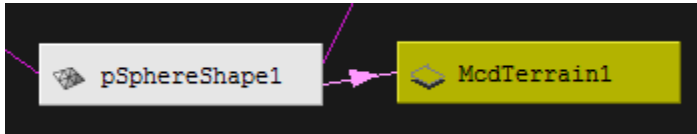
Agent manager can determine which type of agent is activated. System will create contents such as logic decision nodes or action nodes for the active agent and put the contents inside the active agent group.

- **Active**
Active the agent type in this line
- **Modify**
Modify the name of agent
- **Delete**
Delete all the contents of this agent
- **Color buttons**
Select color for active agent, when the agents are placed, the color of bones will be the same color selected here
- **Add Agents**
Add a new type of agent to scene

Terrain Manager

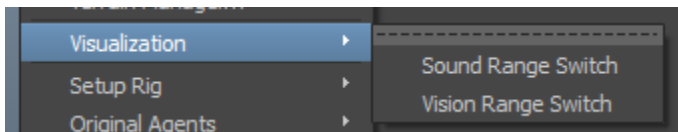


Terrain manager can mark geometry terrain. When the geometry is marked terrain, the agents can interactive with it and the rigid body can collide in it. In fact, the geometry which been marked terrain is connected by a McdTerrain node.

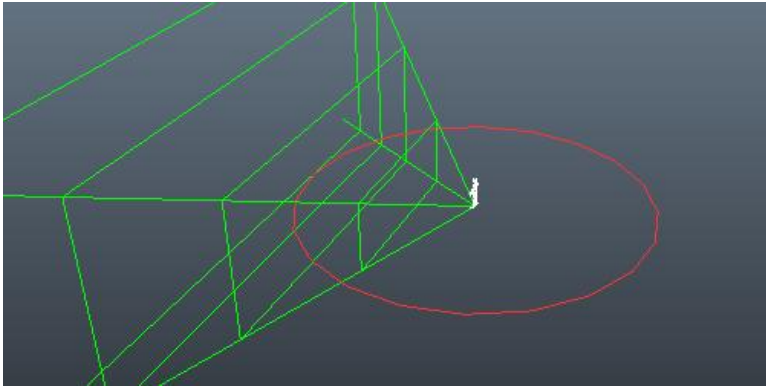


- **Terrain Node**
The McdTerrain node name which connected to terrain geometry
- **Mesh**
Actual mesh name been marked terrain
- **Dynamic Friction**
The friction force when object is moving on terrain
- **Static Friction**
The friction force when object is before moving on terrain
- **Restitution**
The energy of bounces back from collision, increase this make object bounce greater on the terrain
- **Is anim**
Enable this if the terrain has animation or deformation.
- **Is Plane**
Enable this if the terrain is a plane, it can speed up the physical simulation
- **Select**
Select the transform node of this terrain geometry.

Visualization

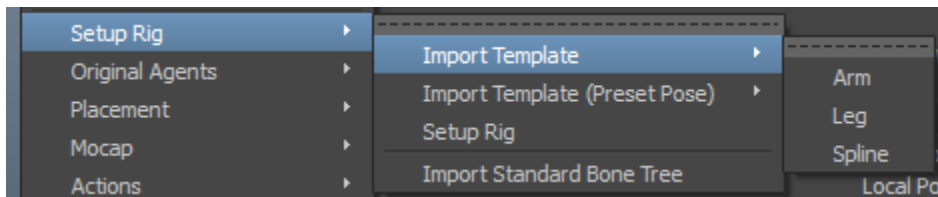


- **Sound range switch**
Switch on/off the sound range
- **Vision range switch**
Switch on/off the sound range



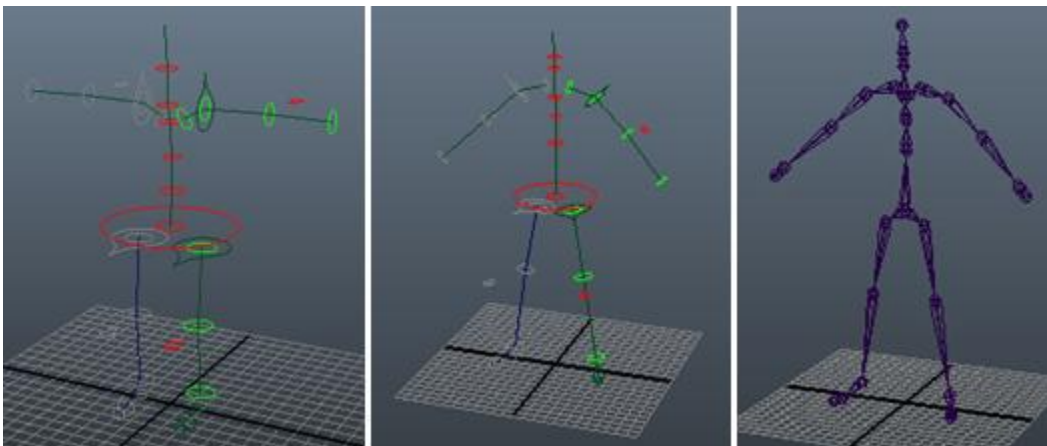
As the picture above, the red circle is sound range, and the green frustum is the vision range.

Setup Rig



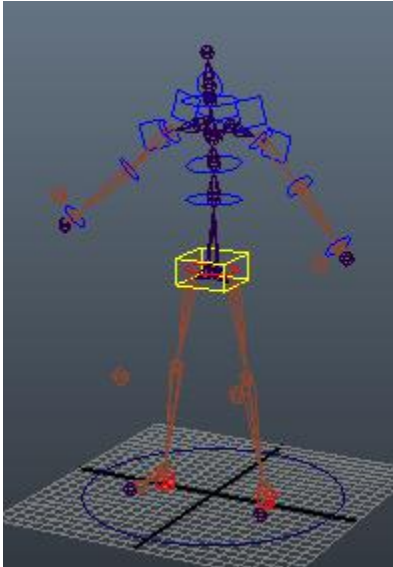
Miarmy provide a free setup pipeline tool, which can generate a simple rig from the template.

- **Import Template**
Import template with default pose
- **Import Template (Preset Pose)**
Import template with the preset pose
- **Setup Rig**
Convert the template to the rig
- **Import Standard Bone Tree**
Import a clean and preset posed bone chain which can be used to motion capture



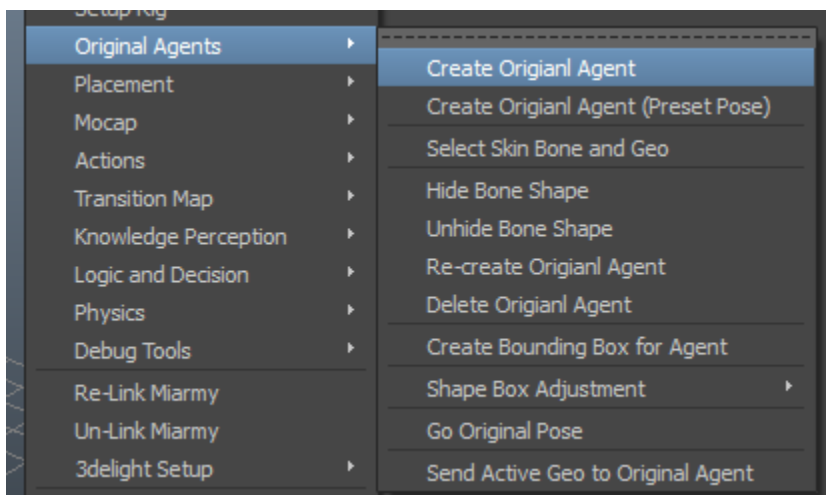
Above 3 pictures, from left to right:

1. Template with default pose
2. Template with preset pose
3. Bone chain with preset pose



After setup, the rig has been generated.

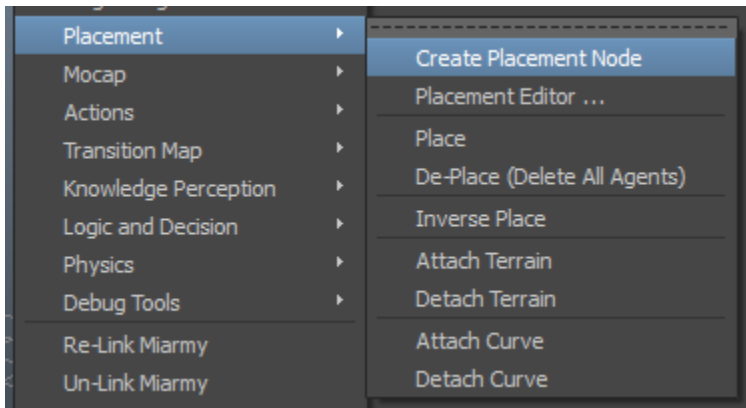
Original Agents



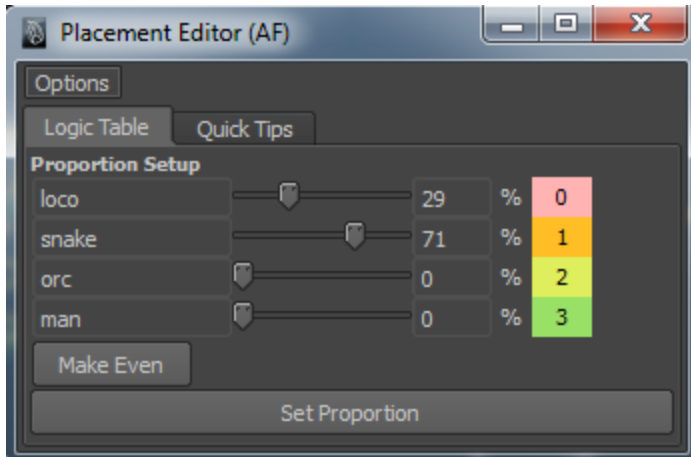
- **Create Original Agent**
Create Original Agent if there is correct rig in Setup group for the active agent type
- **Create Original Agent (Preset Pose)**
Create Original Agent with many preset setup

- **Select Skin Bone and Geo**
Delete all the joints except end ones from current active original agent
- **Hide Bone Shape**
Hide all the box shape of the bone
- **Unhide Bone Shape**
Hide all the box shape of the bone
- **Re-create Original Agent**
Delete the current active original agent and create it again. If you delete some important part from original agent, you can use this tool
- **Delete Original Agent**
Delete the current active original agent
- **Create Bounding Box for Agent**
Create bounding box for current active agent
- **Shape Box Adjustment (This feature is not finished yet)**
- **Go Original Pose**
Clear all rotation value from original agent and put it to the origin
- **Send Active Geo to Original Agent**
Copy geometries and skinning info from setup rig to Original Agent, only can work on active agent type

Placement



- **Create Placement Node**
Create a placement node for you
- **Placement Editor**



Placement editor can setup the proportion for the selected placement node

- **Proportion Setup (slider)**

Only the place-able agent type can be listed here. The agent with original agent is correct agent type. Move any of the slider, the others types will automatically update except 0 proportion type

- **0 proportion**

Crank down to 0 for turning off this type of agent place



- **Make Even**

Auto averaging all the proportion of exist types



- **Place**

Populate agent to scene.

- **De-Place (Delete All Agents)**

Delete all the agents and flush the undo queue. We not recommended delete agents by “del” key because that the agent is actually leave there in memory.

- **Inverse Place**

Create a brand new place node from the selected agents, and this will record follow info:

- Parent node of each agent
- Type of each agent
- Translate and Rotate
- Mute dynamic flag

- **Attach Terrain**

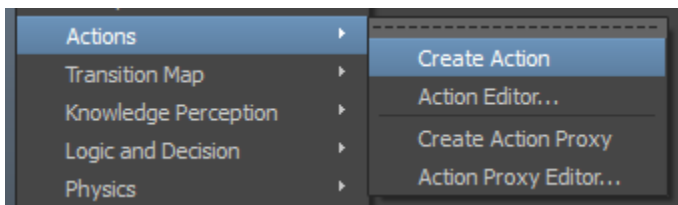
Attach place node to the selected terrain

- **Detach Terrain**
Detach place node from terrain
- **Attach Curve**
Attach place node to the selected curve
- **Detach Curve**
Detach place node from curve

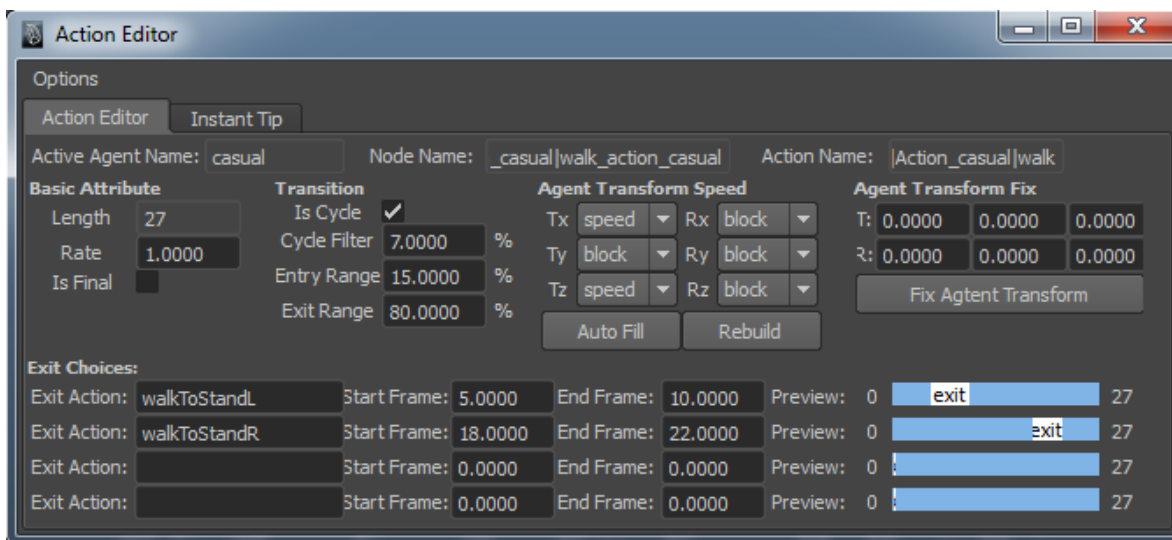
Mocap

- **Import Standard Bone Tree**
Import a clean and preset posed bone chain which can be used to motion capture
- **Human IK (available next time)**
- **Characterize (available next time)**

Actions



- **Create Action**
Create an action node, then playback once and store the animation data to this node, from rig (with animation) to action
- **Action Editor**



- **Basic Attribute**

- **Length**
The length info of current selected action
- **Rate**
The playback speed multiplier, e.g. 2 means playback speed is 2 times than normal
- **Is Final**
Enable this means that once agent transit to this action, never transit out, e.g. dead
- **Transition**
Control the transition between 2 actions
 - **Is Cycle**
Enable: cycle action (e.g. walk), Disable: transition action (e.g. standToWalk)
 - **Cycle Filter**
A percent value stand for a smooth filter when the action performing self-cycle transition
 - **Entry Range**
The range of transition from previous action to the current one
 - **Exit Range**
After this percent range, this action can transit to next action, before that, it playback itself
- **Agent Transform Speed**
The speed of transform node of the agent, this can create the locomotion result, please check out the detail of this in Animation and Action session
 - **Channel(TX, TY, TZ, RX, RY, RZ)**
Block/Open the channel on agent transform node
 - **Auto Fill**
Use the preset channel preset
 - **Rebuild**
Important: After editing the channels, user need click this rebuild button make it works
- **Agent Transform Fix**
This feature can offset the result of transform speed of the root, e.g. one of the motion capture data is x-oriented, but one can use this feature to rotate it back to z-oriented action
 - **Transform Data**
Specify which channel and how many values need offset
 - **Fix Agent Transform**
Once specify the value in blank, click this for performing offset
- **Exit Choices**
In different phase, action can transit to different next actions. We can specify the exit choices here
 - **Exit Action**
The list of actions which the current action will transit to
 - **Start Frame & End Frame**
Between start and end frame, the current action will transit to this exit action if there is a transition signal

- **Preview Bar**

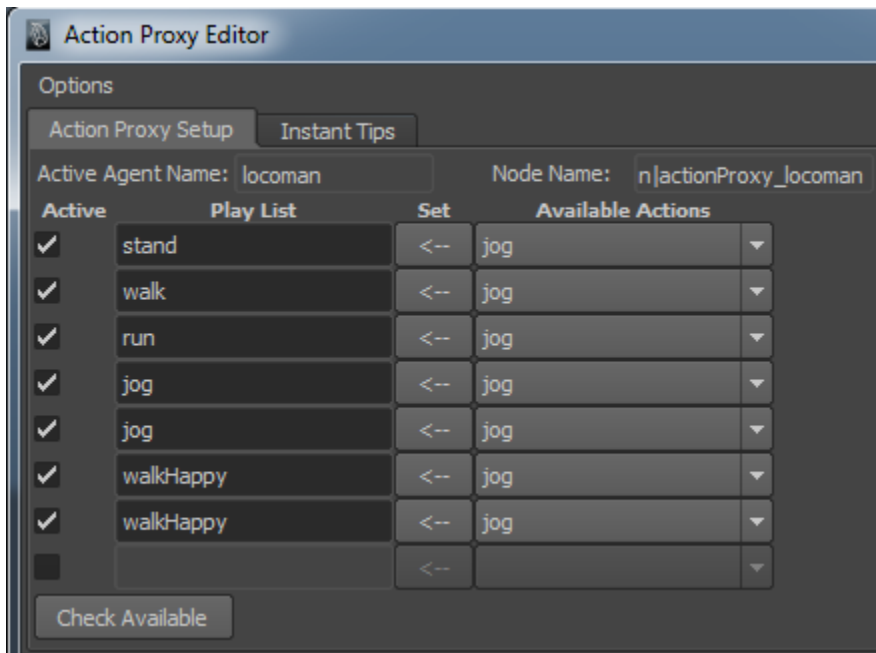
Preview in which phase we can transit to which exit action

- **Create Action Proxy**

Create an action proxy node for current active agent type

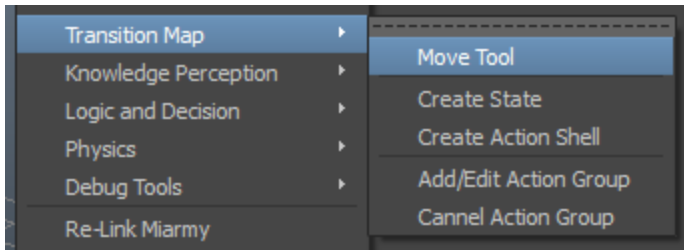
- **Action Proxy Editor**

Action proxy can check the transition between any actions and see is them transit smooth or correct
For more details of action proxy, please check out the Transition session of this manual.



- **Active**
Activate/deactivate element of agent name
- **Play list**
The names of actions in list
- **Set**
From right to left set
- **Available Actions**
Choose which action to be set

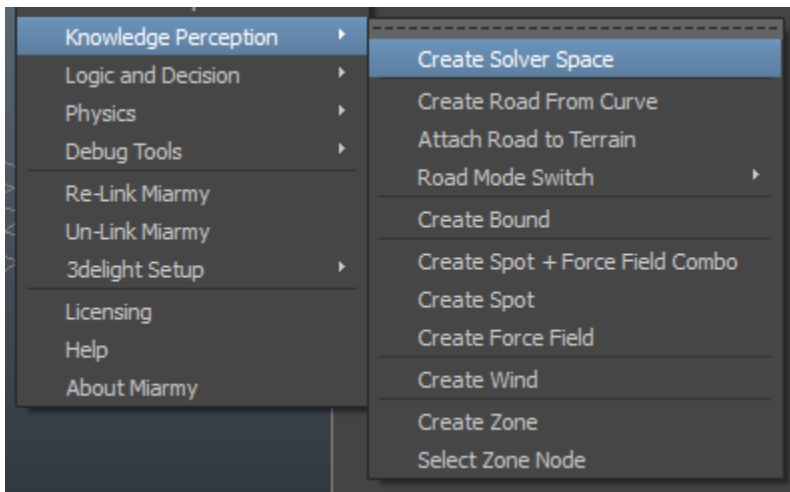
Transition Map



(Note: need switch to the transition map viewport)

- **Move tool**
Click and drag to move the states and action shells in transition map viewport
- **Create State**
Create a new state node for current active agent type
- **Create Action Shell**
Create a new action shell node for current active agent type
- **Add/Edit Action Group**
Add an action group flag to the selected action
- **Cancel Action Group**
Remove the action group flag from the selected action

Knowledge Perception



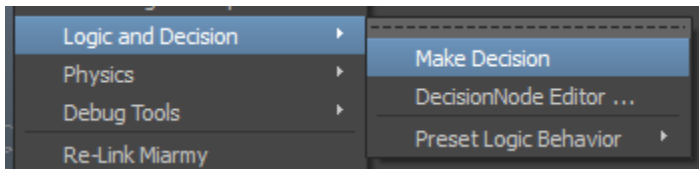
Please check out the channel spec for the detailed information

- **Create Solver Space**
Create a solver space node
- **Create Road from Curve**
Create a road object from your selected curve, the agent can feel it
- **Attach Road to Terrain**

Attach the road to terrain

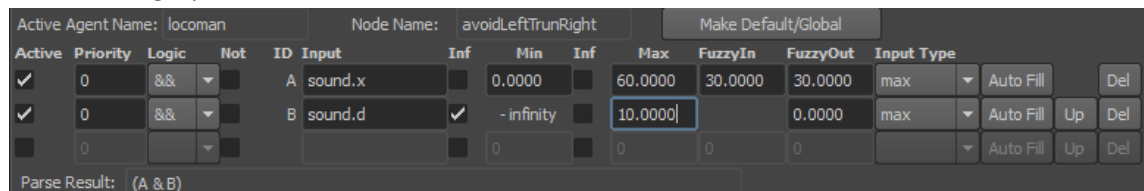
- **Road Mode Switch**
Switch road between flow and road modes
- **Create Bound**
Create bound box/sphere so that the agent can feel it
- **Create Spot + Force Field Combo**
Create a spot node and set the Feel Mode to “both”
- **Create Spot**
Create a spot node and set the Feel Mode to “Only Spot”
- **Create Force Field**
Create a spot node and set the Feel Mode to “Only Field”
- **Create Wind**
Create a wind node so that the agent can feel it
- **Create Zone**
Convert the selected geometry to zone object
- **Select Zone Node**
Select the geometry firstly, and then this tool can help you get the related zone node

Logic and Decision



- **Make Decision**
Create a decision node for the active agent type, for detail, please check out the logic part of this manual
- **Decision Node Editor**
 - **Input Part (Normal Mode)**

The inner logic part



- **Make Default/Global Button**
Change the current selected decision to Global Mode
- **Active**
Activate/Deactivate of this sentence
- **Priority**

The priority of this sentence, bigger value has higher priority

- **Logic**

The operate before this sentence

- **Not**

Determine whether invert the logic result

- **ID**

The Identity of this sentence assigned by the system

- **Input**

Input sentence

- **Inf**

Whether infinity for the fuzzy range in, e.g. "< 10" means negative infinity to 10

- **Min**

The minimum/start value of range in which the result will be true

- **Inf**

Whether infinity for the fuzzy range out, e.g. "> 10" means 10 to positive infinity

- **Max**

The maximum/end value of range in which the result will be true

- **Fuzzy In**

Blur the start of true range, not available when negative infinity

- **Fuzzy Out**

Blur the end of true range, not available when positive infinity

- **Input Type**

Chose the input type between max or average

- **Max**

Take the result which makes this sentence fully activated

- **Average**

Take all the results and calculate out the average, then calculate the activation of this sentence

- **Auto Fill**

Some presets for the user, one can add new preset in McdSentencePresetListGUI.py

- **Move Up**

Change the order between 2 sentences up and down, move current sentence up

- **Del**

Delete current sentence

- **Parse Result**

The result of inner logic

- **Output Part (Normal Mode)**

Active	Decision	Value	
<input checked="" type="checkbox"/>	ry	-120.0000	Auto Fill
<input checked="" type="checkbox"/>	tz	20.0000	Auto Fill

- **Active**

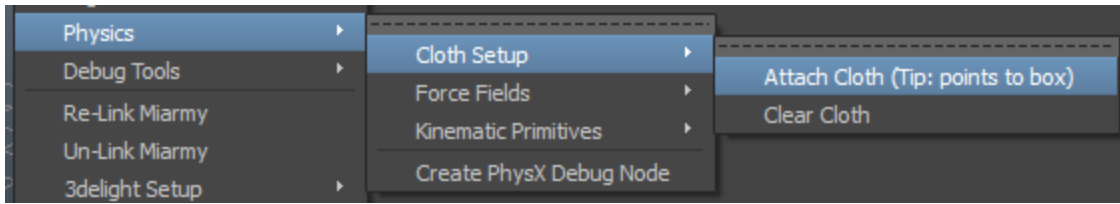
- Activate/deactivate output channel
- **Decision**
The output decision
- **Value**
The output value if full activated
- **Auto Fill**
Some presets for the user, one can add new preset in McdDecisionPresetListGUI.py

○ **Decision (Global Mode)**

Active	Decision	Default Else	Output Type	Defuzz Type	Auto Fill
<input checked="" type="checkbox"/>	ry	-120.0000	Value	Average	Auto Fill
<input checked="" type="checkbox"/>	tz	20.0000	Value	Average	Auto Fill
<input type="checkbox"/>		0.0000			Auto Fill

- **Make Normal Button**
Change the current selected decision to Normal Mode
 - **Default Action**
Play this action when there is no other action activated.
 - **Active**
Activate/Deactivate the default output channel
 - **Decision**
The output channel
 - **Default Else**
System will use this value when the channel are not been activated
 - **Output Type**
The output value change type, absolute value or change rate
 - **Defuzz Type**
The output defuzz type average/blend/max
 - **Auto Fill**
Some presets for the user, one can add new preset in McdDecisionPresetListGUI.py
- **Preset Logic Behavior**
Create the preset logic decision nodes for current active agent type

Physics



- **Cloth Setup**

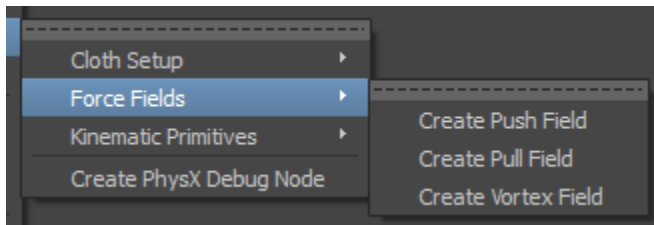
- **Attach Cloth (Tip: points to box)**

Mark cloth for selected geometry and attach the selected points to the bone shape of original agent (firstly select the cloth points, then the bone box shape)

- **Clear Cloth**

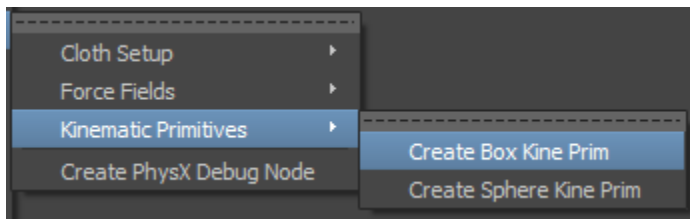
Unmark the cloth for the selected geometry

- **Force Field**



Create PhysX force field

- **Kinematic Primitive**

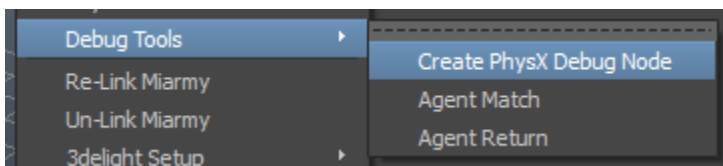


Create PhysX kinematic Primitives

- **Create PhysX Debug Node**

Create a PhysX debug node which can display all the contents of PhysX scene.

Debug Tools



- **Create PhysX Debug Node**

Create a PhysX debug node which can display all the contents of PhysX scene.

- **Agent Match**

Match the original agent to the selected agent, this feature is used to integrate your baking or rendering pipeline

- **Agent Return**

Put the original agent back to origin and default pose

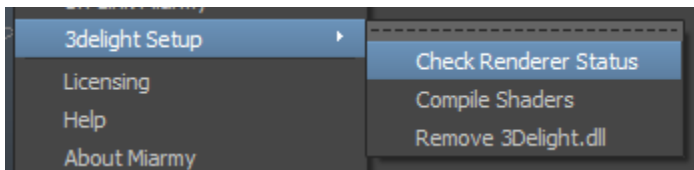
Re-Link Miarmy

Reconnect your Miarmy if you install a new version in the other place

Un-Link Miarmy

Disconnect Miarmy from this Maya

3Delight Setup



- **Check Renderer Status**

Check is your 3delight been installed correctly, check are your 3delight paths been setup correctly, check are your 3delight shaders been compiled ok, etc. If there is anything incorrect, the finished dialog will report them

- **Compile Shaders**

Find the source code of shader in Miarmy installation place and compile using your installed 3delight program, then put the result shaders into 3delight shader path

- **Remove 3delight.dll**

Remove the 3delight.dll from your Maya bin folder. Because system will copy a 3delight.dll to your Maya bin folder when installing, this 3delight will proxy/take over the 3delight of yourself if you already using your own 3delight. Delete the 3delight Miarmy provided here.

Part 3: Agent Infrastructure

Rig

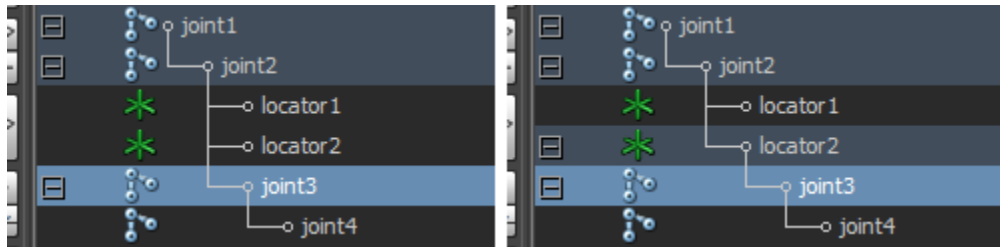
Animation Rig can be anything, FBIK, Human IK, or any kind of custom Maya joint system. Miarmy need the joint structure of your rig for generating the Original Agent and fetch the animation data from that, and nothing else.

Just making sure your bone structure obey some rules and everything will be OK.

Bone structure

Single Chain Joint Structure

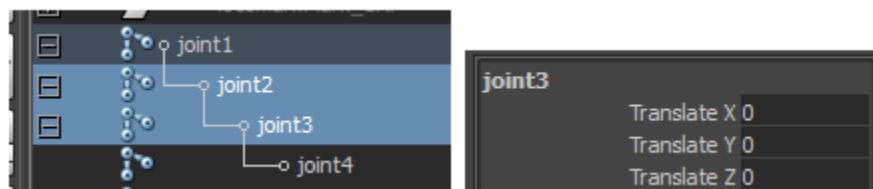
You should make sure your rig is a single chain rigging system. Single chain rigging system means each of the joint (except root) must have a joint parent. No matter how many children nodes under each joint, and no matter how many constrain and IK, just make sure **each bone has a joint parent**. Notice the following 2 pictures, the left one is an example of single bone chain, because each joint (except root) in structure has a joint parent, although joint 2 have several non-joint children. However, the right picture is not a single bone chain. Please notice the parent of joint 3 is locator 2. The locator 2 breaks the chain.



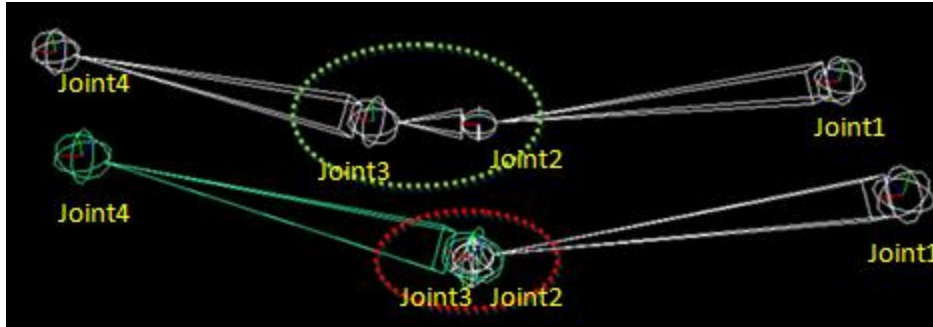
(Left:) single chain rig; (right:) non single chain rig

Bone with length

If you like to enable dynamics on your rig, please make sure each bone has a length, otherwise, once you enable dynamic for the agent originate from this rig, your dynamic system will unstable, even might crash Maya



The joint2 and joint3 overlapped and the joint2 doesn't have length



(Green:) correct joint2 with length (red:) joint2 and joint3 overlapped without any length

Bone Naming

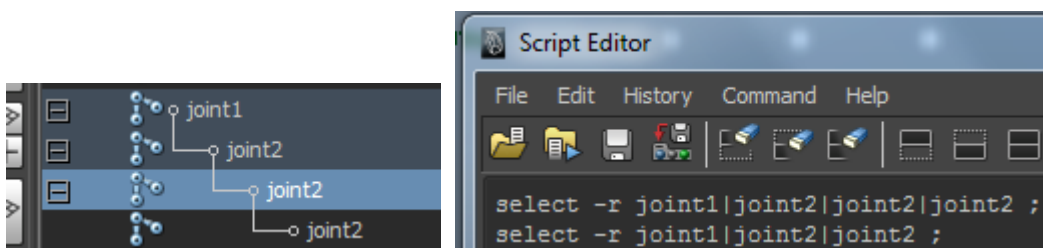
We highly recommend you make the name of the bone unique because the repeated name may bring some unpredictable problems. Non-unique name is no problem in most case but we are not sure is there further problem in features of coming future.

Please take a look at the following 2 pictures, the left one contains some joints which name is not unique whereas the right bones are all unique name.



(Left:) joint name non unique; (right:) joint name unique

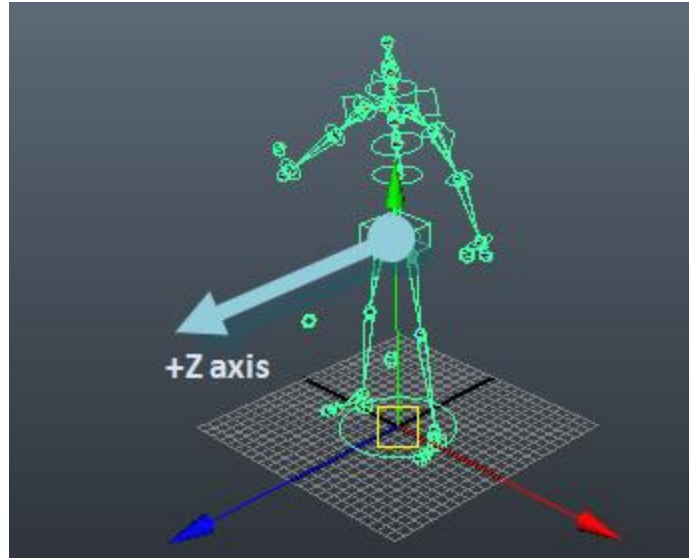
And once you select a joint with non-unique name, the verbose in script editor will be its full dag path instead of its node name.



Select the non-unique name of joint

Joint Orientations

The root joint should orient to positive Z axis, because all of Miarmy engine algorithms (like the sound, vision, field and so on) and action system take the +Z axis of agent as the face front. Please make sure your rig and animation is facing to +Z axis.



Facing to +Z axis

About binding

The geometries on the rig will not join the render pipeline, you need duplicate and bind the geometries to the original agent (and we have script can do this for you automatically). We will talk about this in following chapters

Original Agent

What is Original Agent? And why we need it?

Original agent is the blueprint or template of agents. It is generated from animation rig. Original agent contains a bunch of Miarmy necessary information we need them to populate agents. Each one of Original Agent is responsible for populating one type of agent.

You may ask, if we have ability to populate agents directly from animation rig, why we need Original Agent. The answer is obvious:

- Animation rig is complex and contains many extra useless nodes and information for building agents. These stuffs will make Maya scene messy.
- Original agent has much extra necessary information and nodes when populating agents. We don't want to add these stuffs directly to your animation rig.

So, the animation rig is not part of your Miarmy scene, usually we just need it to be referenced to scene.

Original agent is a main and important part of your Miarmy scene. It contains the same bone structure information of the animation rig, contains enough necessary information to populate agents. Additionally, it is very “clean and clear”

Creating Original Agent

For creating original agent, you need firstly make sure that your animation rig meets the requirements we mentioned above and then just simply put you entire rig into the “Setup_<AgentName>”. And you can create original agent with menu item Miarmy > Original Agent > Create Original Agent.

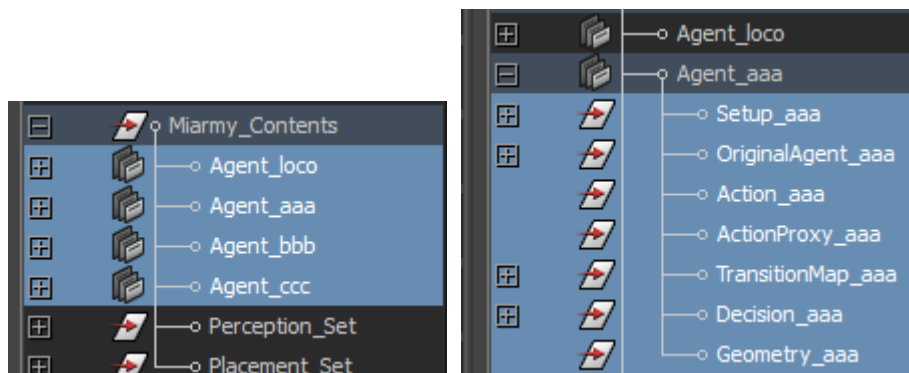
There is another situation we need notice. Some rigging system has control joints structure and deformer joint structure, please make sure your deformer joint structure above the control joints. Or you can make the

Once you click “Create Original Agent”, our code will parse the structure from your animation rig and create a new rig based on your rig. After that, it will add a bounding box for each joint of the new rig. The new created

From another point of view, if you original agent can be created correctly and successfully, that prove that your animation rig is ok.

Agent Group Node and Agent Type

The agent group nodes is the root node for managing all the contents of one agent type. Our system distinguishes different agent types using this group node. You can add, edit or delete the agent group in Miarmy > Agent Manager. Please take a look at the follow example, there are 4 types of agent in scene, each one of them have its own setup, original agent and other contents. And in each one of agent type node, there are several groups nodes for storing the contents of this agent type.

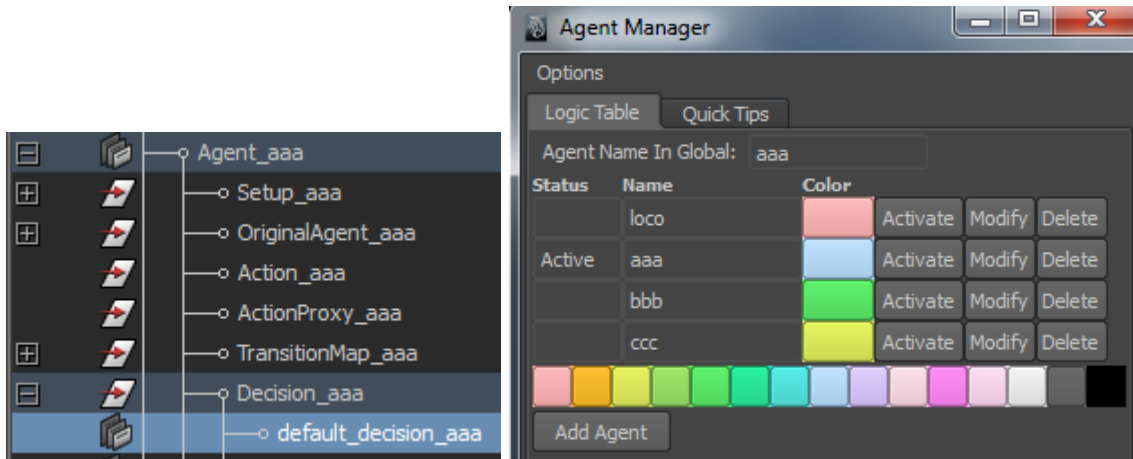


(left:) Agent group nodes; (right:) contents in Agent_aaa

Technically, the agent group node is a simply transform node and the Maya type name is “McdAgentGroup”.

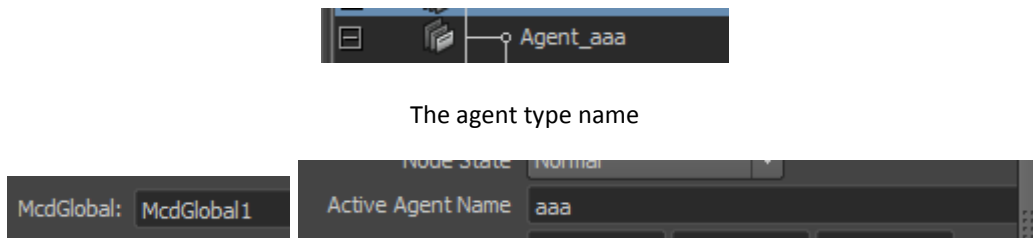
Due to that usually there is not only one type of agent in scene, when we want to create some contents for specific agent type, we need specify which type is activated for current been dealing with, and the new created

stuff will automatically belongs to the active type. For example, we want to create a logical decision node for type “aaa”. We have to activate the “aaa” type of agent in Agent Manager. Then, the system will know the new created decision node belongs to “aaa” agent type and group the decision node to the Decision_aaa node. For detail, see “Menu Item, Agent Manager” chapter.



(left:) new created node belongs to aaa; (right:) Agent Manager, agent which type name is “aaa” has been activated

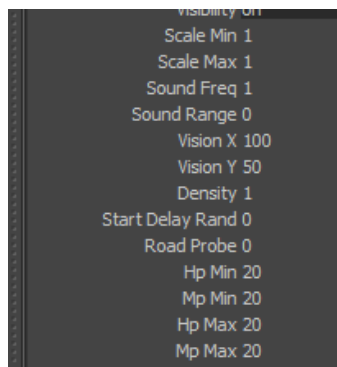
The active agent name will be stored in McdGlobal node, with attribute name is “activeAgentName”, the following picture demonstrate this.



The agent type name

The attribute in McdGlobal Node

Importantly, there are several attributes on each agent group node, and these attributes belong to this type of agents



Extra attributes on McdAgentGroup node

- **Scale min/max:** random range for sizing agents
- **Sound Freq:** the sound frequency of agents
- **Sound Range:** the sound range (amplitude) of agents. If 0, it will automatically use the diagonal length of bound box of this agent
- **Vision X/Y:** vision range frustum angle in horizontal and vertical
- **Density:** the bone density for physical simulation
- **Start Delay Rand:** some random frame before starting action
- **Road Probe:** how far the agent can feel in front road
- **HP/MP Min/Max:** random range for setting initial HP/MP

Render Geometry Binding

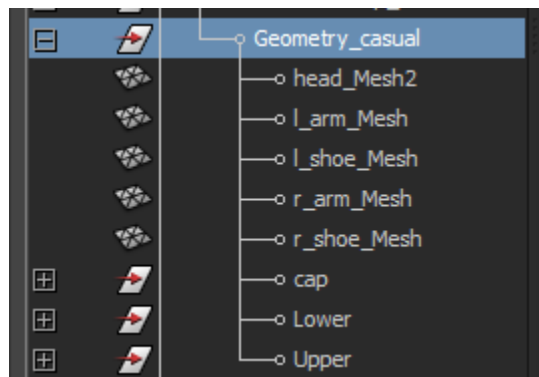
Different render pipeline support different binding method.

- If you are going to use 3delight build-in plugin for rendering, you can use arbitrary binding method, such as smooth skinning bind, rigid skinning bind, even constrain.
- If you are going to use Mesh Drive Pipeline for rendering you scene directly in Maya or export geometry cache, you need to use smooth bind skin the geometries to the original agent. Notice it's **only** support smooth skinning bind.

About render, see details in **Part 8 Rendering**.

Render Geometry Management

After skinning, you need put your geometries to Geometries_<agentNam> group like this:



Putting the geometries of original agent in Geometry_<agentName>

Cloth geometry should be put in his group.

Render Geometry Shader

Different render pipeline support different kind shader on geometries.

- If you are going to use 3delight build-in plugin for rendering, you can only use the default shader like Lamber, Blinn, or Phong. Or you can create custom surface or displacement shader with McdRMShader and McdRMDispMat. And our plugin doesn't support bump map, we only support displacement map.
- If you are going to use Mesh Drive Pipeline for rendering you scene directly in Maya or export geometry cache, any type of shader is available such as mental ray, v-ray, and so on.

About render, see details in **Part 8 Rendering**.

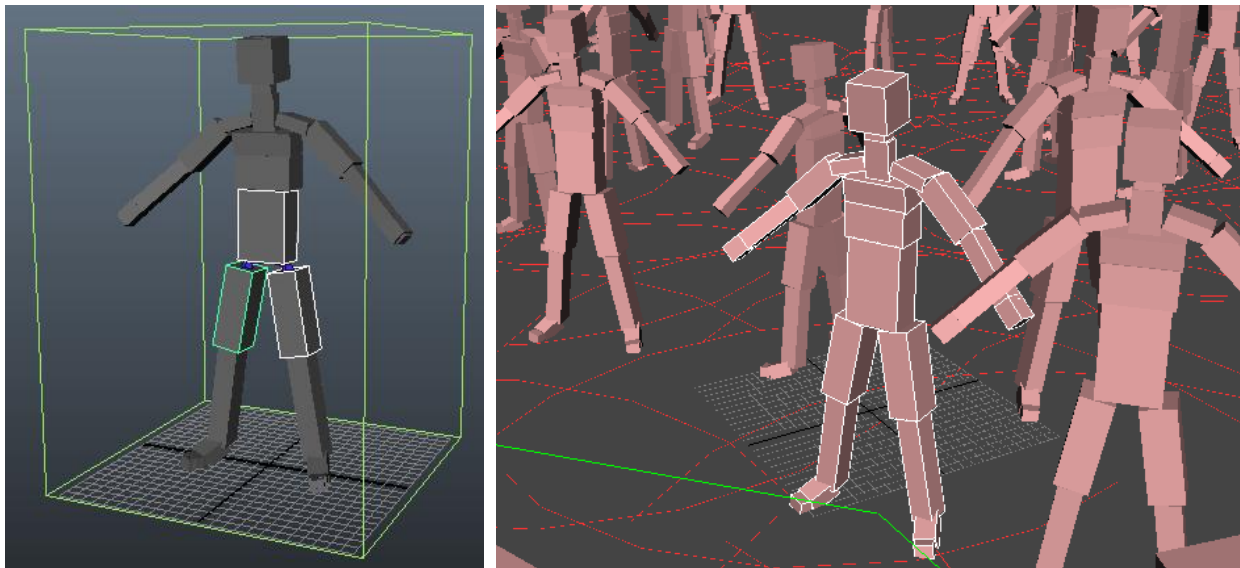
Bone Shape for Display & Physical Simulation

Adjust the boxes of the original agent and make them look right size and transform. The placed agents will be the same as the original agent.

Also, don't forget resize the green bounding box, make it can bound your agent. No need precise, but roughly bound the agent. This bounding box can determine:

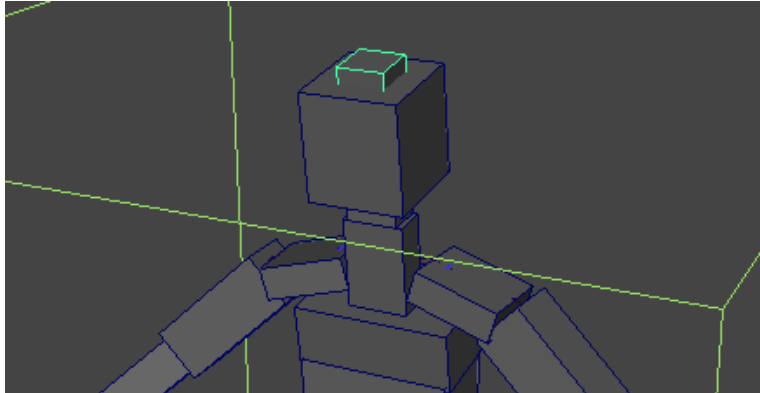
- Default sound range
- OpenGL display culling (The bounding box of each agent)
- Split render culling

Be careful: don't scale or move the joint!!



(left:) the original agent boxes; (right:) the placed agents

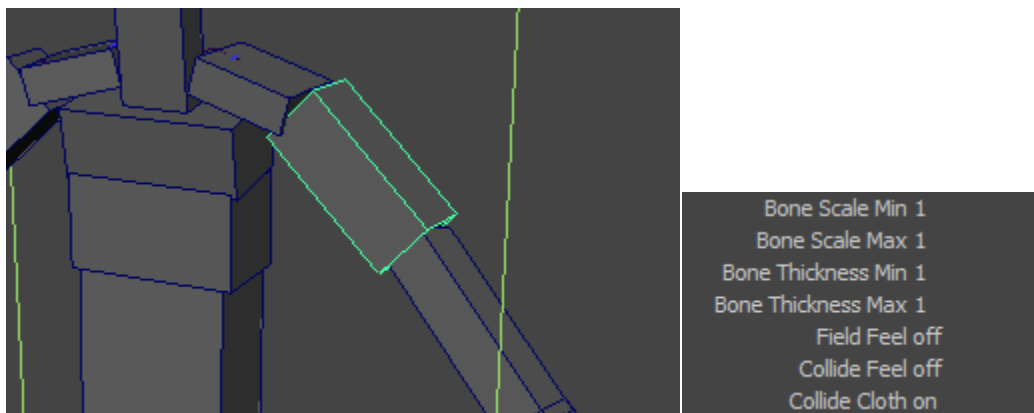
The final bone (end box) is not necessary to deal with, just ignore it.



You can ignore the head end. It's useless and will not join the animation and physical simulation

Bone Attribute & Flags

Also, each of the boxes has some extra attributes can be applied:

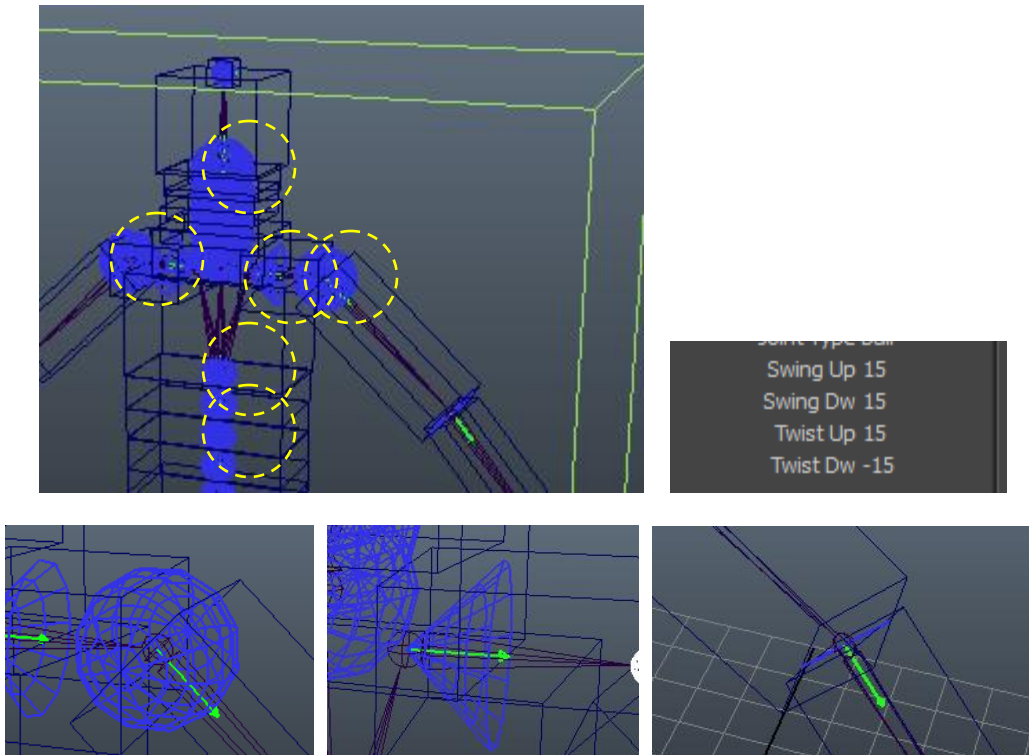


- **Bone Scale Min and bone scale max** determine the current bone random size when populating agents
- **Bone Thickness** is not available for current version of Miarmy
- **Field Feel** determine whether this bone can feel Maya field after turning on dynamics
- **Collide Feel** determine whether this bone can join collision checking when we apply “collide” channel
- **Collide Cloth** determine whether this bone can collide against with cloth

We will call them **flagged bones** when collision detection or interacting with force field.

Physical Joint Type

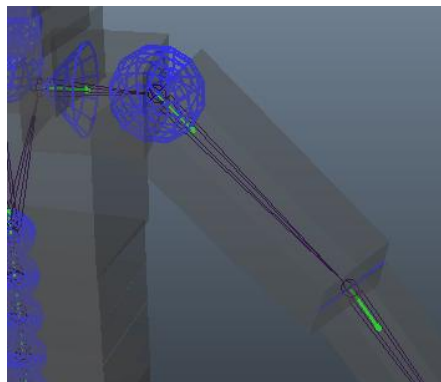
Between each pair of bones, there is a joint. It is use to build rag doll joint. Each joint can be one of 3 types like the picture below middle.



3 different types of joint respectively: ball, shoulder, hinge

Physical Joint Direction

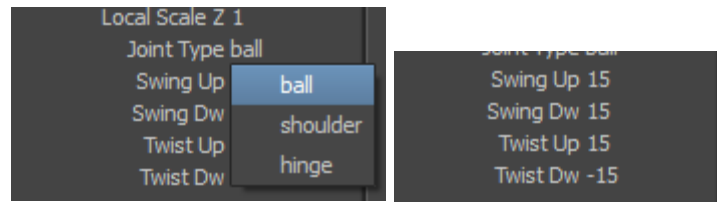
Each joint have a green arrow, you should always point this arrow to the next bone



Pointing the green arrow to next bone

Physical Joint Limits

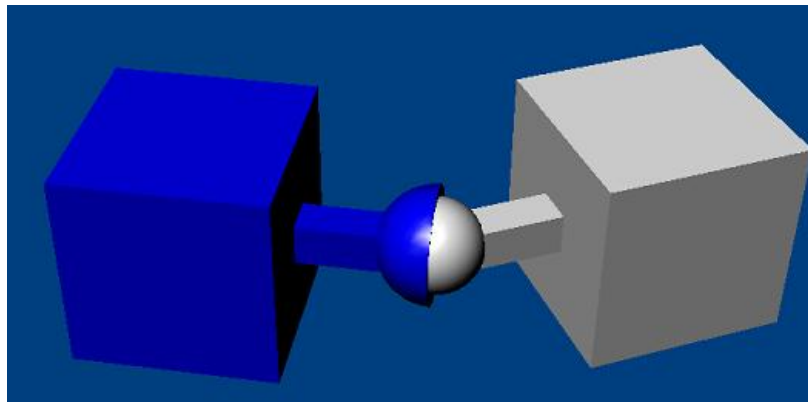
Each of joint has some rotation limits need to setup, especially on elbow and leg. And the same attribute name maybe have different meaning if the different joint type. The swing attribute cannot be the negative value.



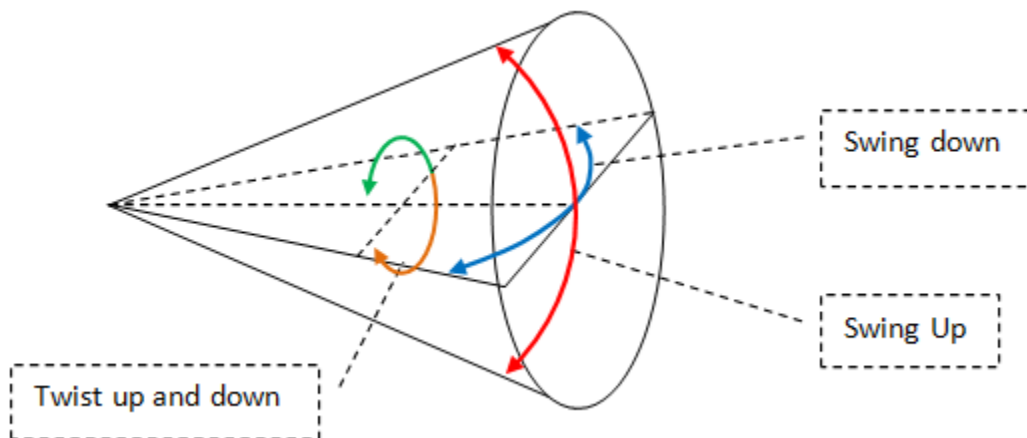
Joint types and limit attributes

Ball joint

Ball joint can swing and twist:

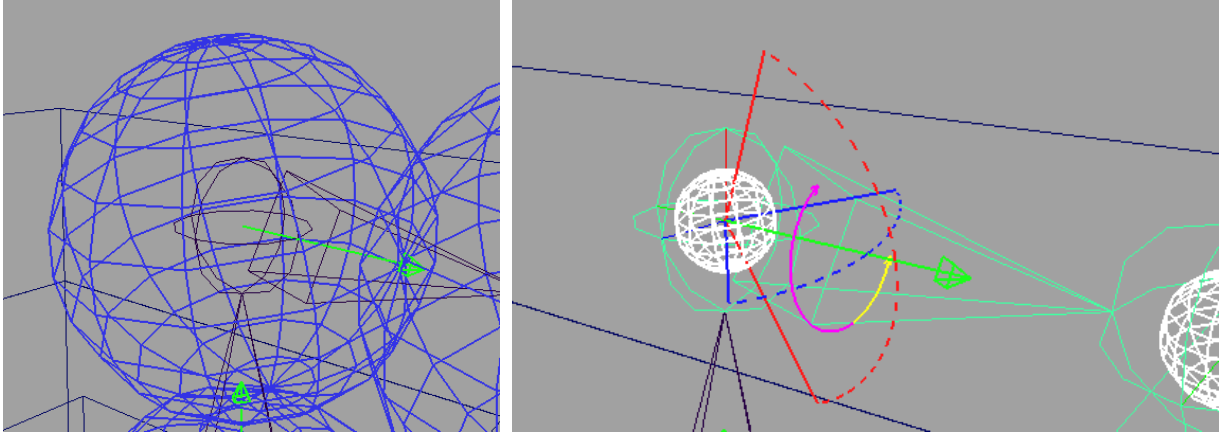


Ball joint example



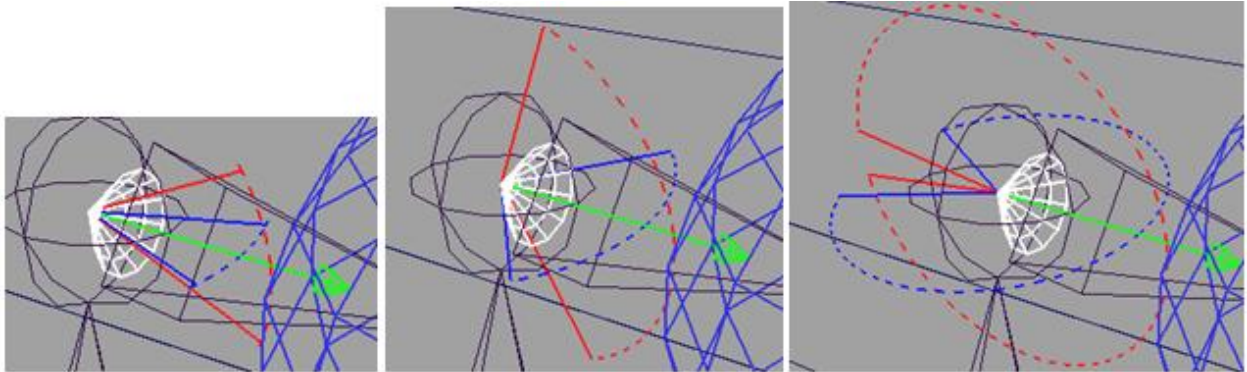
Ball joint swing and twist limit

After Miarmy 1.5, once you select the physical joint, it will become to a visible manipulator



Left non-selected, Right Selected

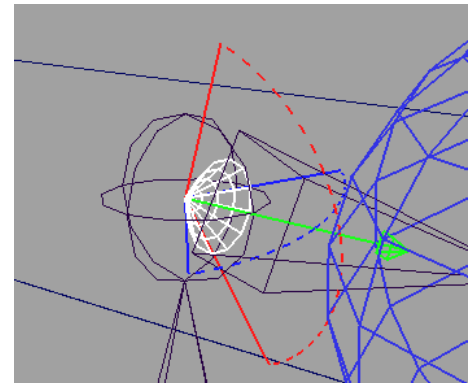
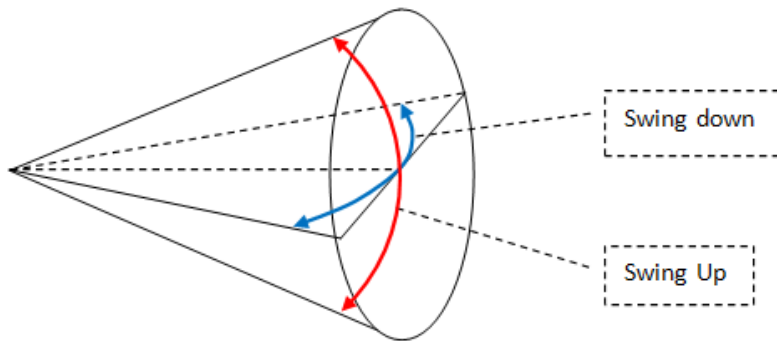
The ball joint setup (Blue: swing down, red: swing up, yellow and purple: twist in counter/clockwise)



(From left to right) Small range of rotate, middle range or rotate, wide range of rotate

Shoulder joint

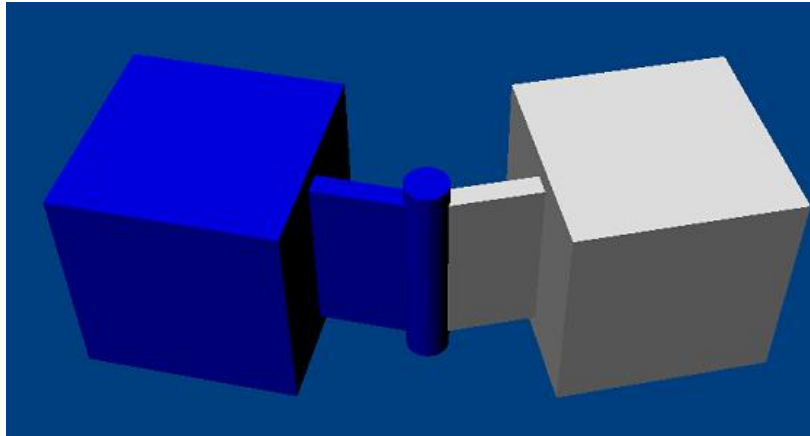
Shoulder joint can swing only, cannot twist



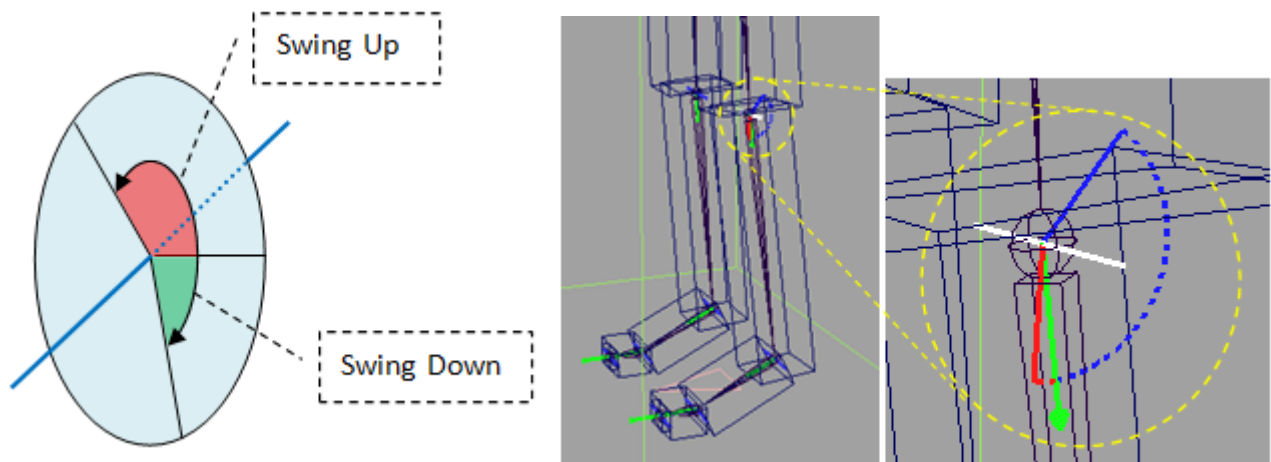
Shoulder joint limit, swing up and down limit

Hinge joint

Hinge joint can rotate as Z axis only:



Hinge example



Hinge joint rotation limit (the right one: can be rotate back easy but cannot rotate to forward)

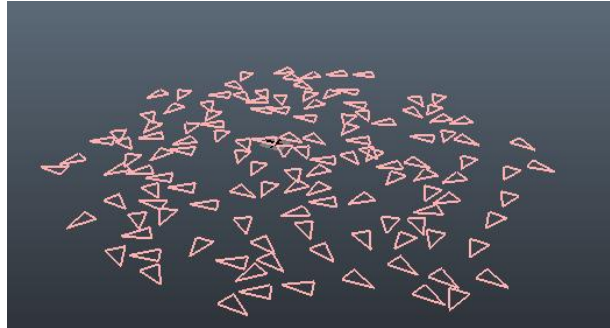
Placement

Place node is a tool can populate your agent into the scene. There are some main features:

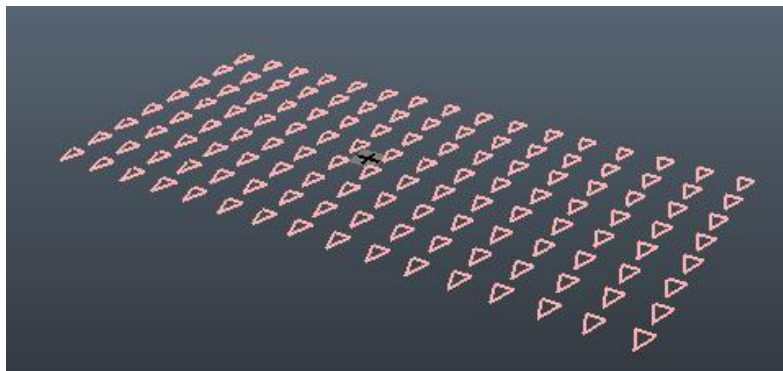
- You can adjust proportion for different types of agents
- Automatically check overlap if all place nodes in the same parent
- You can parent the place node to a specific node for achieving “Hierarchical Placement”
- If you make the place type “custom”, you can use MEL or Python to adjust the inner data structure of it.
- Inverse placement mechanism can store “the information of your custom placement scheme” to placement node. From your agents to placement node.

Point & Formation Placement

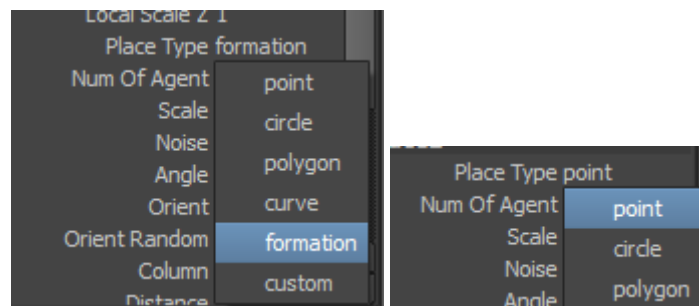
Firstly, we are going to introduce you 2 independent placement types “point” and “formation”. Please look at the picture below.



Point placement type



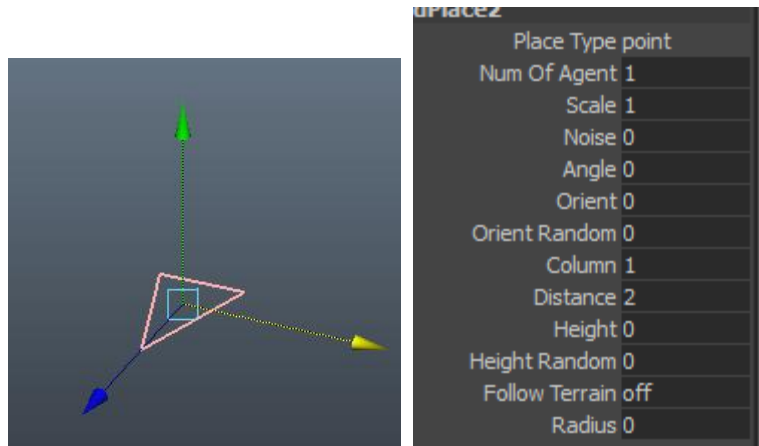
Formation placement type



Note: in current version, the **circle** and **polygon** type is not available.

Creating Place Node

For create a placement node, simply click Miarmy > Placement > Create Placement Node, The default type of place node is “point”:



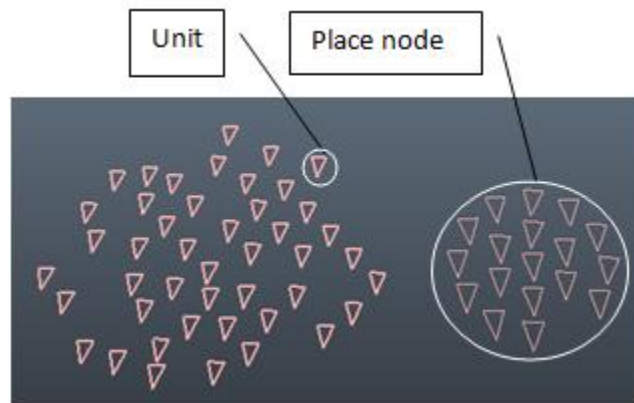
(left) default initial place node (right) attributes on place node

Terminology

The single unit of triangle stand for one agent instance, we call it “Place Unit”.

The entire placement group we call it “Place Node”

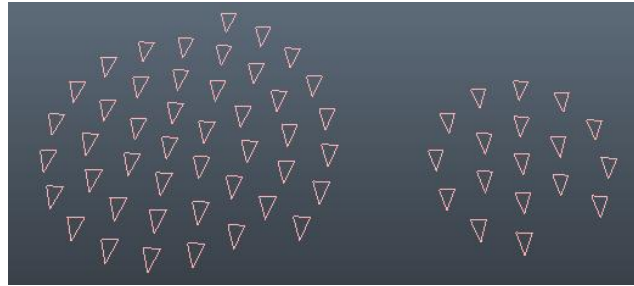
The process we generate agents from place node, we call it “placement”



Attributes on Place Node

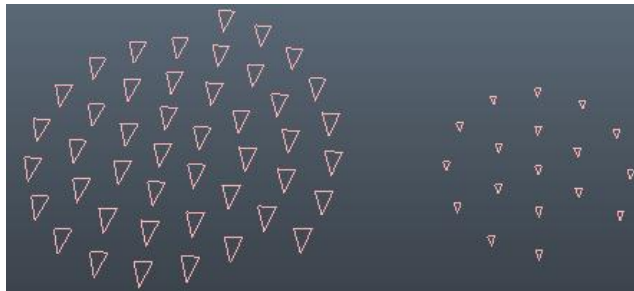
There are some attributes on each of place node. And different placement type need different attributes, let's take a look at the following example:

Num of Agent:



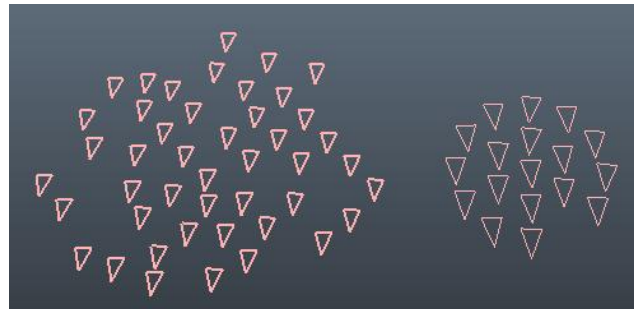
The “numOfAgent” determines how many agents to be placed

Scale:



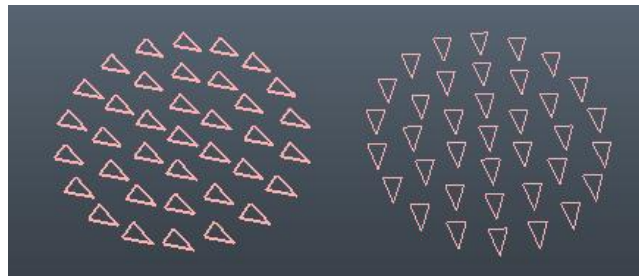
The “scale” determines the size of single place unit

Noise:



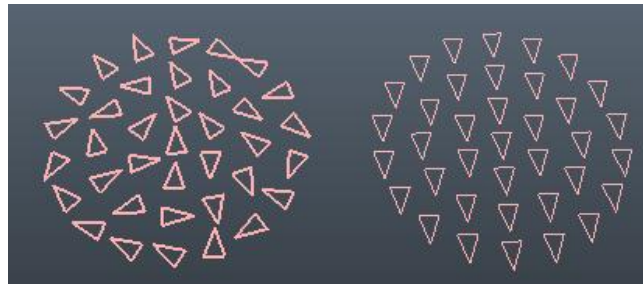
The “noise” determines the random position on X and Z axis of each place unit

Orient:



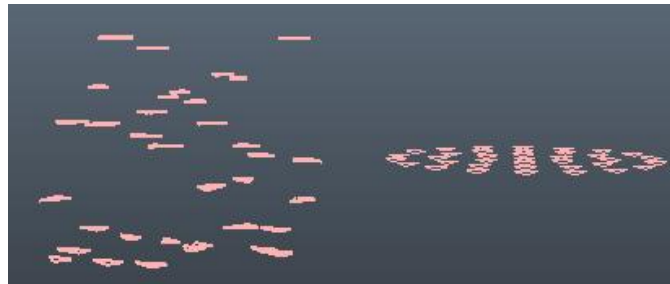
The “orient” determines the orient of every place unit uniformly

Orient Random:



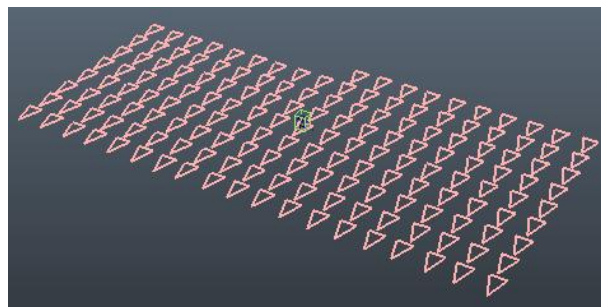
The “orientRandom” determines the random orient of each single place unit

High Random:



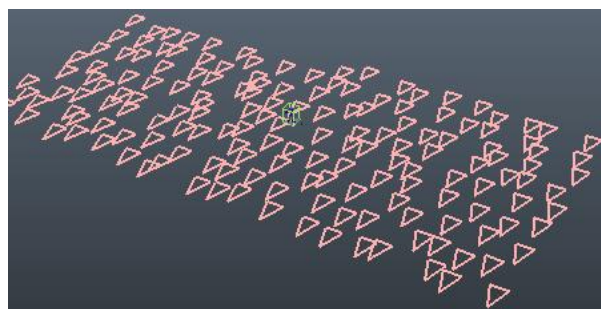
The “heightRandom” determines the random of Y-axis of each place unit

For “formation” placement:



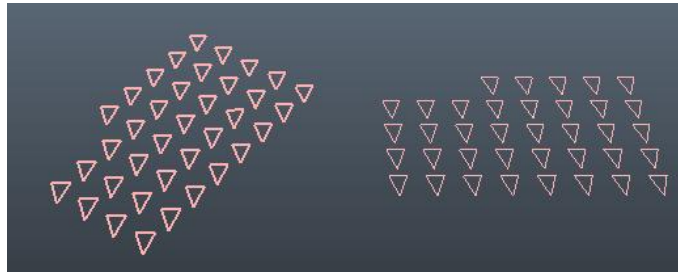
The place type is “formation”

Noise:



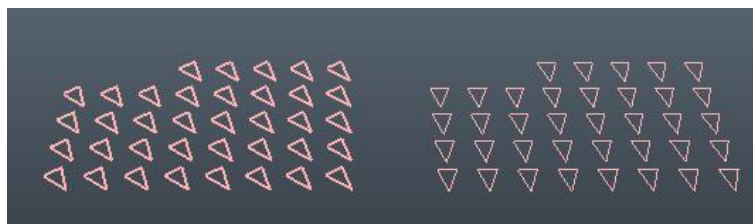
The “noise” working on the formation place type

Angle:



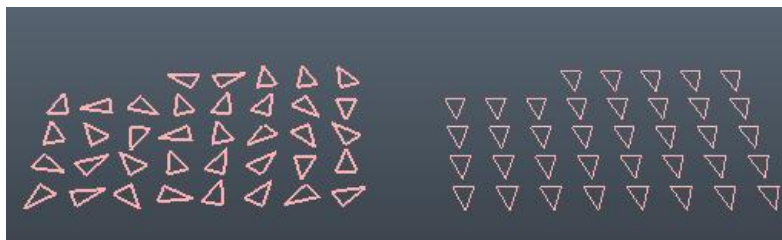
The “angle” determine the entire rotation for formation place type

Orient:



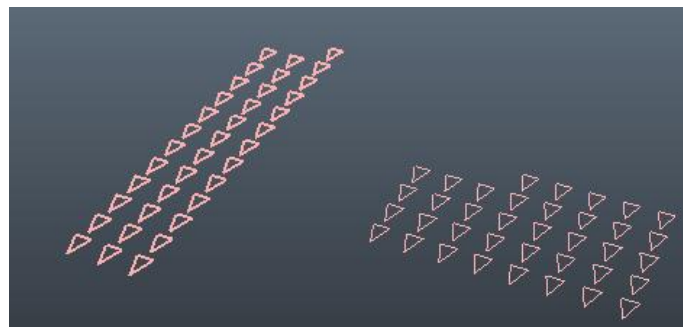
The “orient” determines each single rotate uniformly

Orient Random:



The “orientRandom” determines the random orient of each single place unit

Column:



The “column” determines how many columns for a formation placement

The summary of place node attributes:

- **Place Type:**
 - **Point:** arrange based beehive
 - **Circle:** not available for current version
 - **Polygon:** not available for current version
 - **Curve:** arrange follow curve
 - **Formation:** arrange by row and column
 - **Custom:** arrange by use
- **Num of Agent:** Number of agent for current node
- **Scale:** unit size
- **Noise:** random x, z value of unit in space
- **Angle:** (only work for formation place) the rotate of entire place node
- **Orient:** unit rotation Y
- **Orient Random:** unit rotation Y random from 0 to 360
- **Column:** (only work for formation place) column number of a group of unit
- **Distance:** distance between units
- **Height:** place node height, translate Y
- **Height Random:** unit translate Y random

Populate Agents from Pace Node

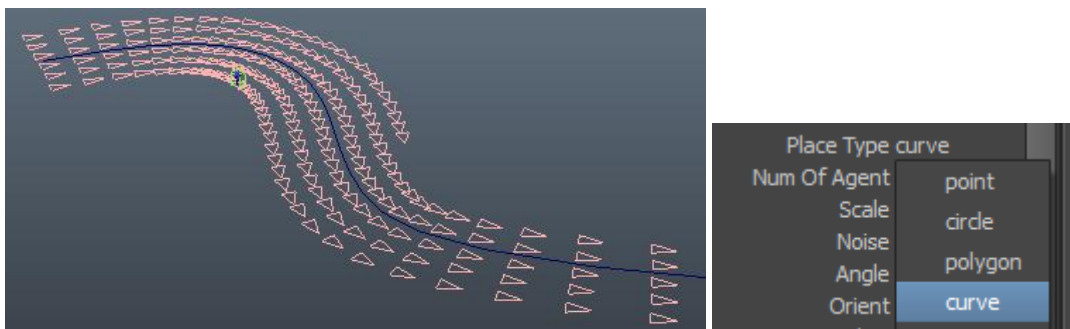
Then, after you adjusting place node, you can place agent by Miarmy > Placement > Place

Or you can delete all placed agent by Miarmy > Placement > De-Place (Delete All Agents)

The reasons we provide the “De-place” function for you is not only we need delete all agents automatically, but also clear/flush the memory. As everybody knows that, Maya has a feature called “undo”. It means if undo feature enable, the object you delete is not actually deleted, all the data still there. It’s a hazard in crowd simulation since it will take huge memory, as well as, the physical stuffs are still there in scene.

So please use “De-place” as much as possible instead of “delete manually”.

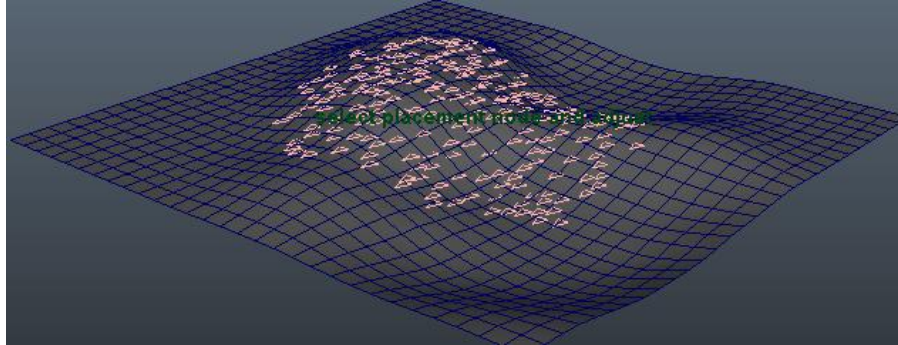
Curve placement



(left:) Place node attach curve (right:) need to switch to “curve” mode

You need first select place node, then select the curve, click Miarmy > Placement > Attach Curve

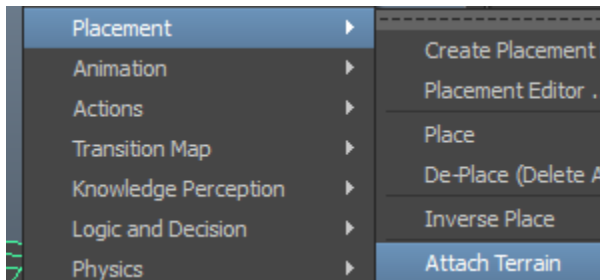
Terrain Attachment + Point placement



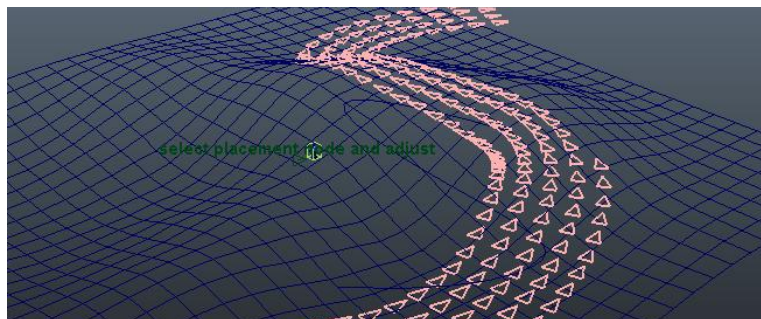
Terrain attach is happened in Y direction of unit (place element). That is means no matter what type of placement you are using, they can be attached to terrain.

For attaching terrain:

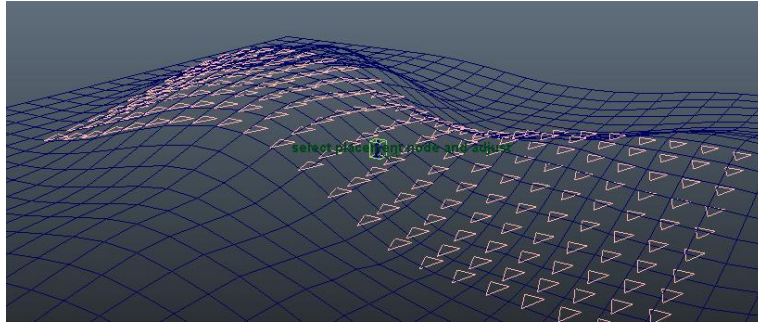
1. First make sure a plane-like geometry is zero transformation (clear all translate, rotate, scale values).
2. Select place node firstly, and geometry secondly, then click



Combining with place type, curve and terrain, you can apply any kind of placement scheme.

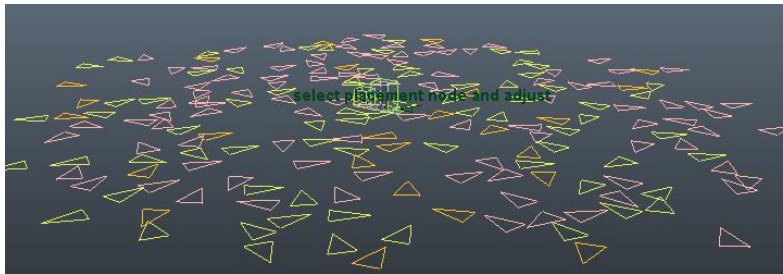


Terrain Attachment + Curve Placement



Terrain Attachment + Formation Placement

Proportion in Place Node

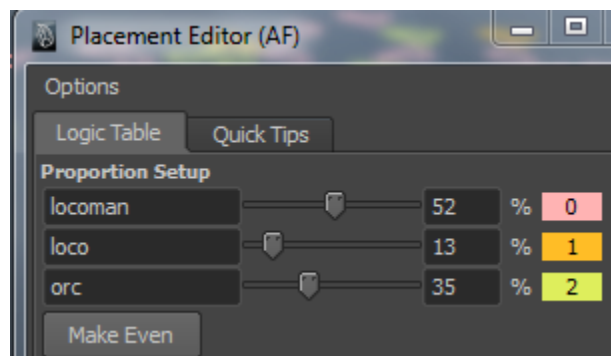


Please take a few minutes to recall the “Agent Manager”, “Original Agent” and “Multiply Types of Agent”. If there are **more than one type** of agent in scene and all of them **have “original agent”**. The 2 conditions should both meet. We can assign proportion for the placement node:

Open placement editor:

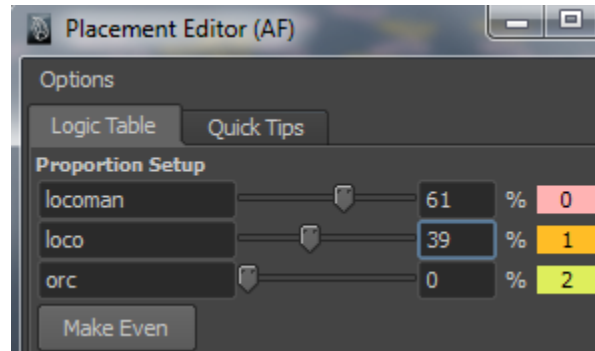


Select your placement node and this window will update automatically



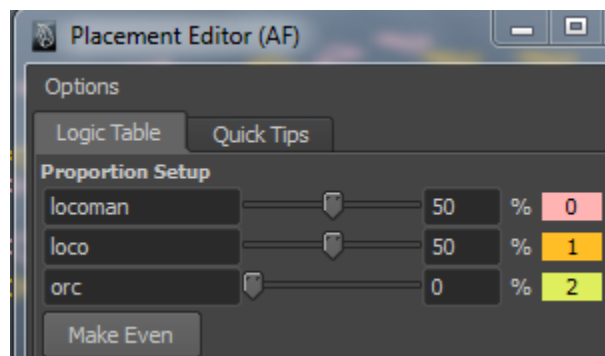
The proportion Editor

Simply tweak the slider for assigning proportion to your placement node. The total proportion is always sum to 100%. If you don't want to assign proportion for specific agent type, just simply crank it to "0" for locking. Notice the "orc" agent has been locked.



Lock down the orc agent type

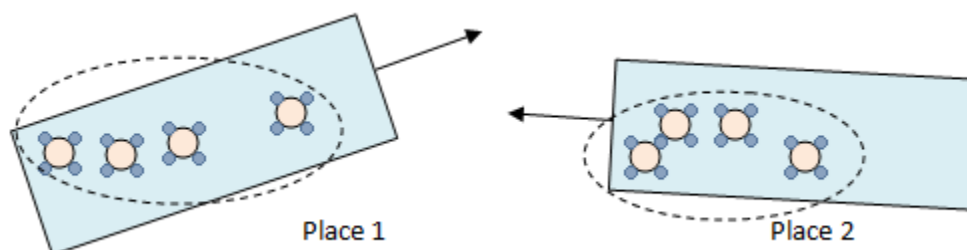
Click "make even" for averaging the assignment. (Notice the orc has been locked)



Make even for non-locked agent types "locoman" and "loco"

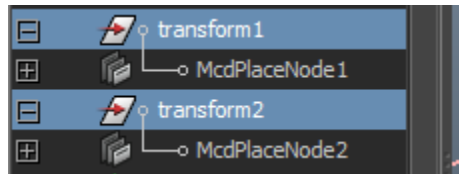
Hierarchical placement

Hierarchical placement means that you can place your agent in hierarchical local space instead of world space. For example, you have several battleships in scene, and each ship has some agents on the deck. These agents are walking and doing something, the ships are moving. At this time the agents on deck will naturally moving with ships also. If we put our place node under ship transform node which means a child node of ship node, the agents populated from this node will automatically be the children of ship. These agents will act in their parent's local space respectively.

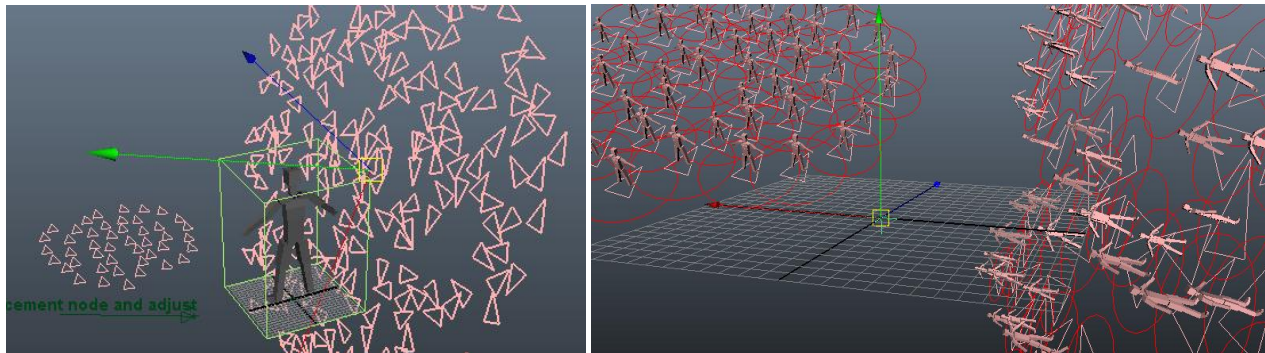


Agents are acting on moving ships

For achieving this, the only thing you need to do is parent your place node to its parent. Like this:



Once you place your agents, they will automatically host to the same parent of place node.



Hierarchical placement, (left) before placing; (right) after placing

Solver Space

Technically, take a look at the picture above. They (2 groups of agents) are only separated by transformation space but not solver space. Because Miarmy take local coordinate to solve, the 2 groups of agents may affect each other even not in same world position. (Different world positions maybe the same local position)

To separate them totally and never affect each other, we need solver space node. Later, we will talk about the solver space more detail.



The pictures above show the different parenting scheme. The left shows agents separated by transformation space, whereas the right one shows agents separated not only by transformation space but solver space.

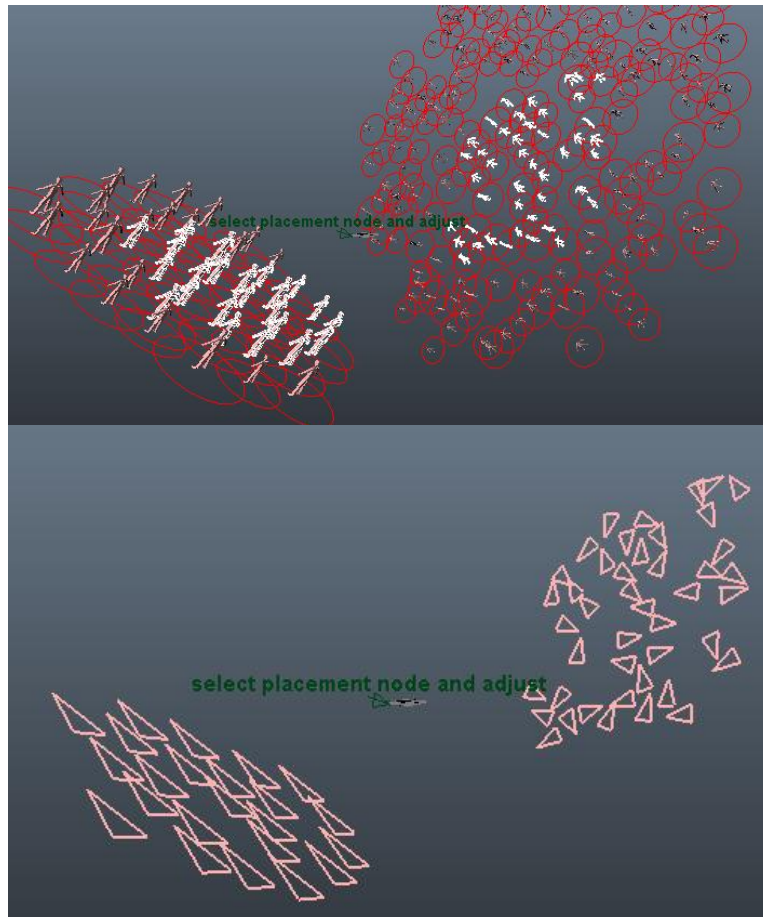
Inverse Placement

Inverse placement is a process which from the selected agents, create new place node. The new created node will record the information inherit from the selected agents.

The inherited information includes:

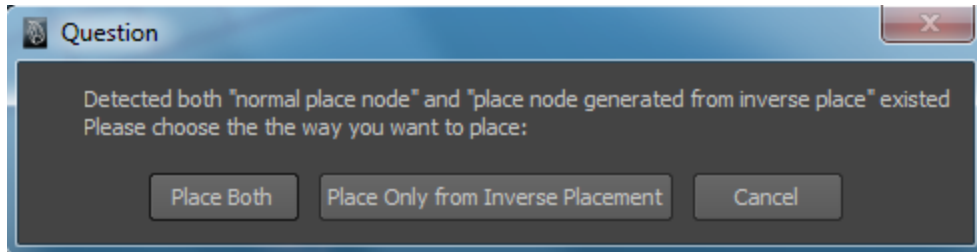
- Translate of each place unit
- Rotate of each place unit
- Type ID of each place unit
- Parent of each place unit

Usually pipeline is, firstly you arrange agents by your hands (move them, rotate them, parent them), just like a tactical arrangement, and then select some agents, apply inverse placement by Miarmy > Placement > Inverse Place.

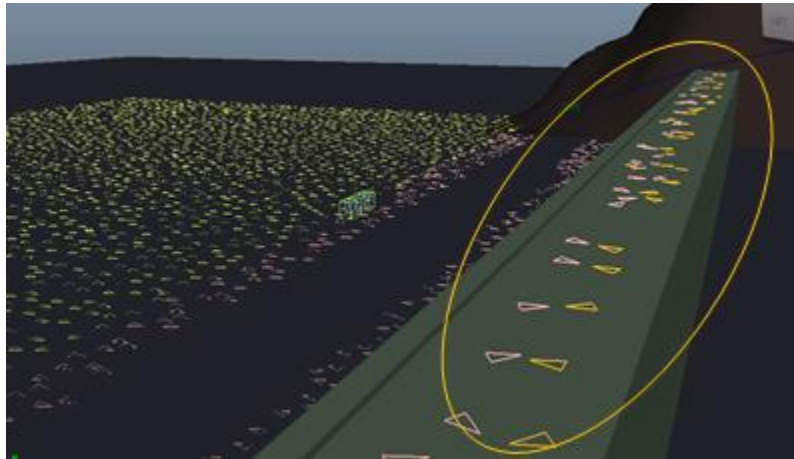


(up :) select some agents, (down:) from selections inversely create new place node

Please Notice Important: There may be some overlapped placement slot in scene after inverse placement. We highly recommend you clear old place node after inverse place node generating. If you place agent by place node and inverse place node at the same time, the system will prompt you for confirm.



You can use inverse placement to place agent according to your rehearsal just like the picture below:



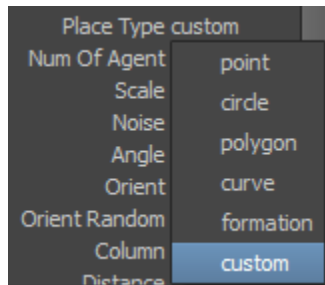
Inverse placement result

You may notice the orange cycle placement are arranged by inverse placement

Place Node Data Structure

You can arrange the placement by procedural methods. Using MEL or Python script, you can access and modify the placement data easily and directly.

We should setup the place type to “custom” for a place node firstly:



Switch to “custom mode

Transformation Info

`<place node>.placement[<unit id>].agentPlace[<infomation>]`

- <place node>: the shape node name of place node
- <unit id>: the Nth agent unit
- <information>: from 0 to 7 stand for:
 0. Agent Type ID
 1. Translate X
 2. Translate Y
 3. Translate Z
 4. Rotate X
 5. Rotate Y
 6. Rotate Z

Example:

```
1 mc.getAttr ("McdPlace2.placement[0].agentPlace[0]")
2 mc.setAttr ("McdPlace2.placement[0].agentPlace[1]", 10)
```

Line1: get the agent type from unit 0 of McdPlace2 node

Line2: set the translate X to unit 0 of McdPlace2 node

Parent Info

You can also specify each agent parent:

<place node>.parentSet[<unit id>]

Example:

```
3 mc.setAttr ("McdPlace2.parentSet[0]", "locator1", type = "string")
```

Line3: set the parent for unit 0 of McdPlace2 node

Real Example

You can refer it from the placeStadium.txt in the <Miarmy Directory>/samples (the same place store the samples)

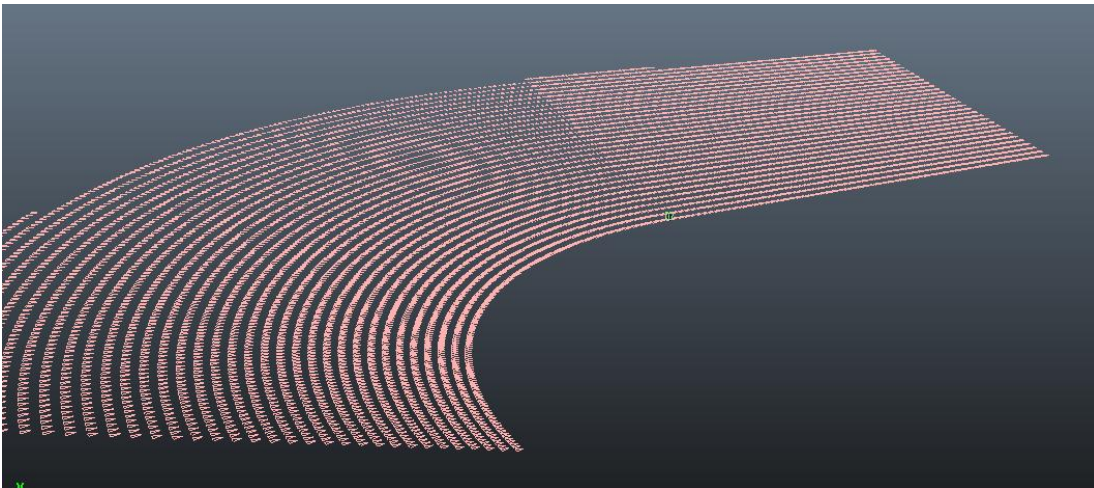
This code can place stadium:

```

1 import math
2 import maya.cmds as mc
3 for i in range(5000):
4     col = i % 150 # x
5     row = i / 150 # y
6     stri = str(i)
7     mc.setAttr("McdPlace3.placement[" + stri + "].agentPlace[1]", col * 20) # x
8     mc.setAttr("McdPlace3.placement[" + stri + "].agentPlace[2]", row * 15) # y
9     mc.setAttr("McdPlace3.placement[" + stri + "].agentPlace[3]", -1 * row * 50) # z
10
11 for i in range(5000):
12     col = i % 150 # x
13     row = i / 150 # y
14     theta = float(col) / 150.0 * 1.5708 # half PI
15     stri = str(i)
16     rInit = 1800
17     rIncrease = 50.0
18     posX = (rInit + rIncrease * row) * math.cos(theta)
19     posZ = (rInit + rIncrease * row) * math.sin(theta)
20     mc.setAttr("McdPlace2.placement[" + stri + "].agentPlace[1]", -1 * posX) # x
21     mc.setAttr("McdPlace2.placement[" + stri + "].agentPlace[2]", row * 15) # y
22     mc.setAttr("McdPlace2.placement[" + stri + "].agentPlace[3]", -1 * posZ + rInit) # z
23     mc.setAttr("McdPlace2.placement[" + stri + "].agentPlace[5]", -1 * theta * 57.2958 + 90) # ty

```

After executing the code in above, the result will be:



The result of stadium placement

Animation and Action

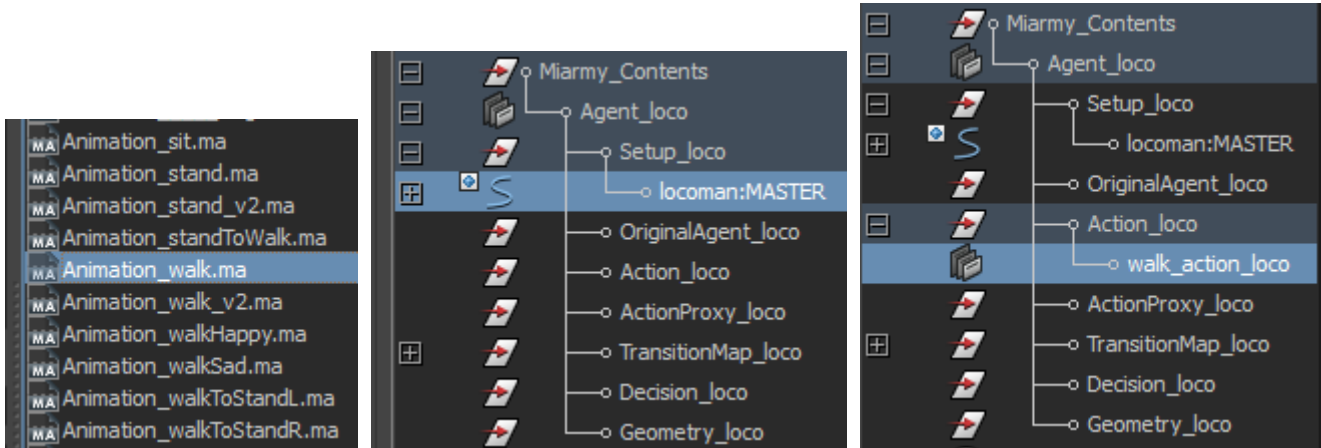
Action is one of most important infrastructure for build behavioral crowd animation. We create actions nodes from animation on the rig.

Note: here we just introduce the creation process, for usage and details, please refer the **Part 6 Action**

Animation to Action Pipeline Suggestion

When you creating action node, we suggestion you take the following pipeline. It would be easily for management and further modify. Note: maybe you have better solutions, here just a suggestion.

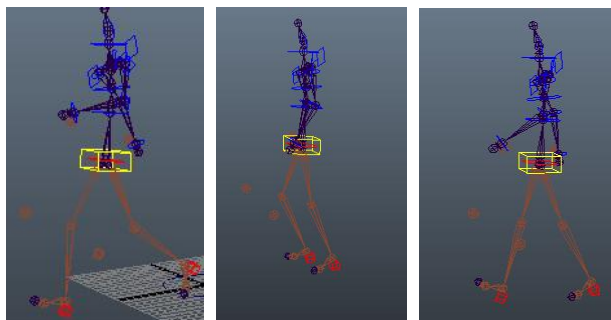
Please take a look at the pictures below. Firstly, we suggest save as each animation file to single file (the first picture). In each file, there is a rig (better be referenced into scene) under the Setup_<agent> name node (second picture). You can freely key frame animation or import animation to this rig. Finally, in each file, there is only one action node and you can export this node to anywhere.



The action create pipeline

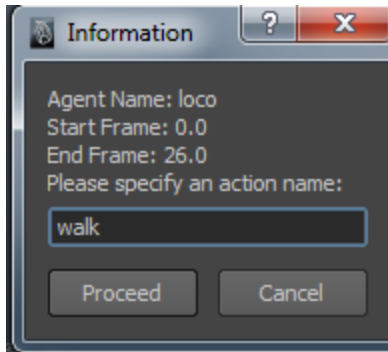
All in all, each file contains a **single** piece of animation and **only one** action node. It would be easily for management.

Create Action Node



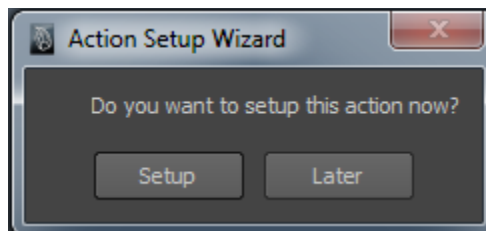
Animation on the rig

The action node will be generated for active agent type. The only thing we are going to do is click Miarmy > Actions > Create Action. And you will see some prompt need you fill or confirm:

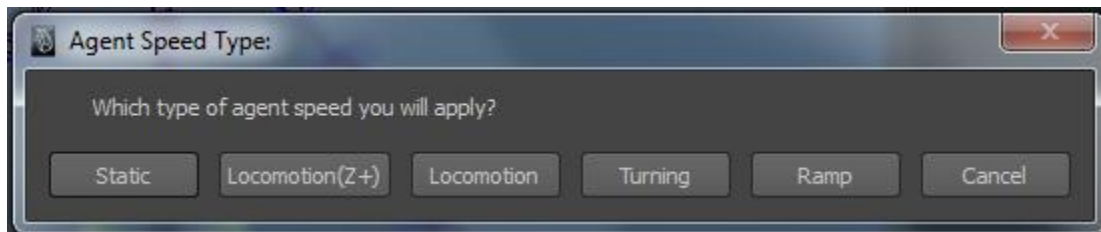


Specify an action name

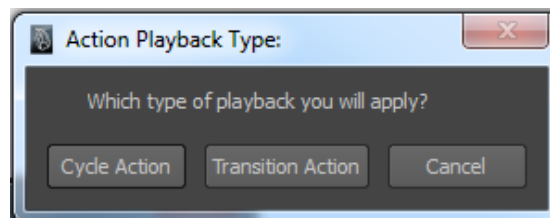
The action setup wizard will pop out for setting your action node up:



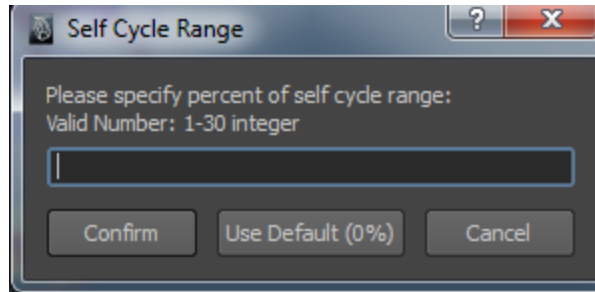
If you click Setup, the wizard will begin.



- **Static:** agent stands without transform change (e.g. stand, sit, cheer)
- **Locomotion(Z+):** agent walks in z direction straight (e.g. walk, stand to walk, jog, run)
- **Locomotion:** agent walks in horizontal space (e.g. walk to left)
- **Turning:** agent turning to left or right (turn left, turn right)
- **Ramp:** agent go up or go down (go upstairs, climbing)



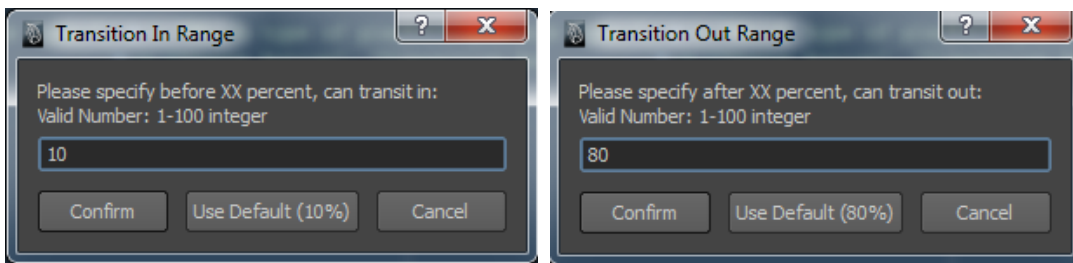
- **Cycle:** self-cycle animation (e.g. walk, stand, run)
- **Transition Action:** middle action between 2 cycle action (e.g. stand to walk, walk to run)



If you select cycle action, there is an additional attribute “cycle range”, means the self-transition range, more specifically, if the action is go into the last XX%, the cycle will occur, self-transition. It blends the end self-transition part to the head part of the action by some percent.



As picture above, after playing back green area, the action will perform self-cycle.



Either you select “cycle action” or “transition action”. There will be 2 attributes need set up, “transition in” and “transition out”.

Transition in:

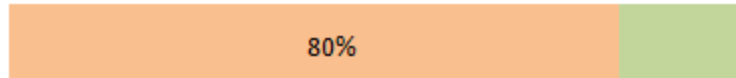
When previous action transit to current action, some percent should be smooth blended. This is the entry range of this current action. Like the picture below, when the previous action transiting to current action, 10% of action length should be blended from previous end to current start.



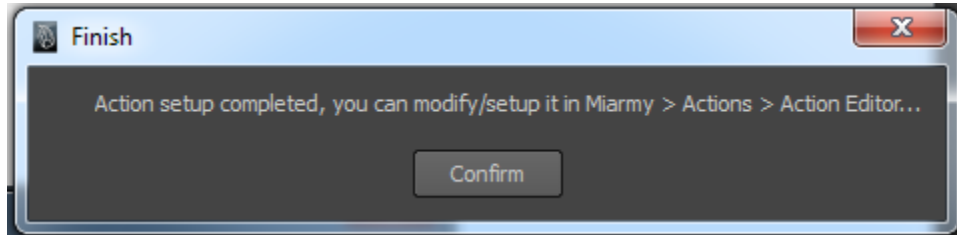
When transiting in, blend with previous action

Transition Out:

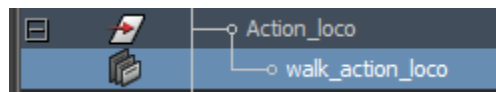
When current action should transit to next action, only when the current action playback after exit range percent, the current action can transit to next action, otherwise, maintains the self-playback



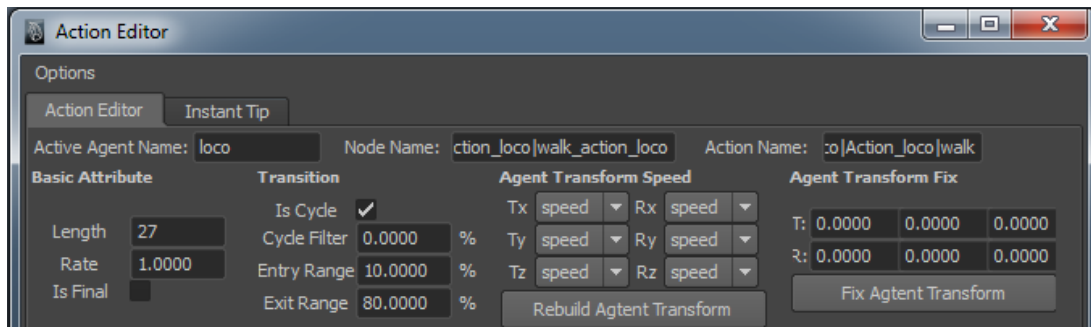
Only after 80% playing back, the current action can transit to other actions



Your action nodes will be created in:



You can select one of it and open Action Editor for checking or editing details:



Action editor

The more detail about action and something under the hood will be described in **Part 6 Action**

Transition Map Building

Concept

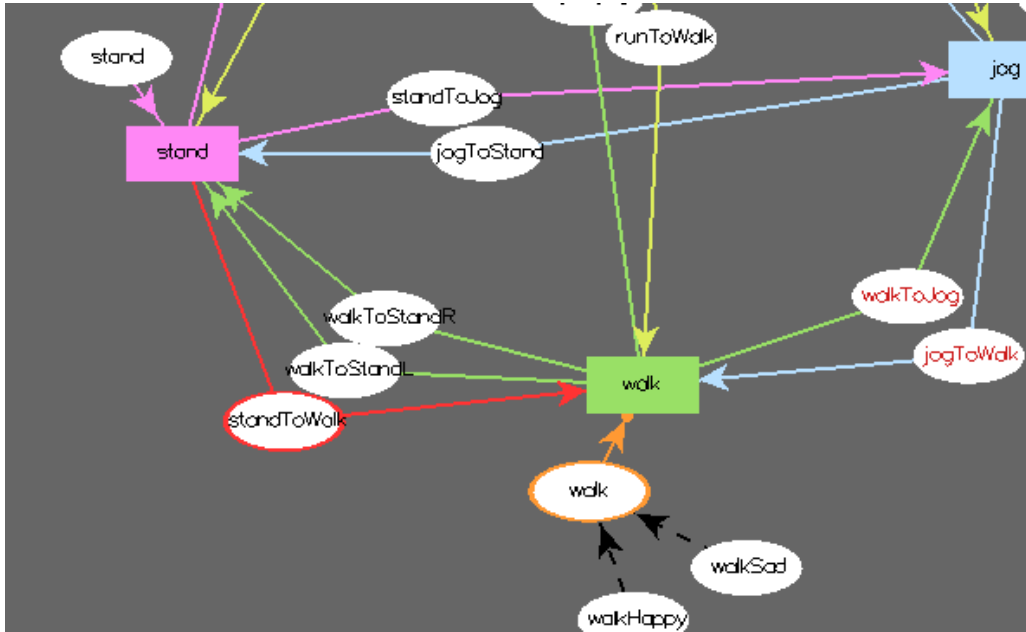
Action transit from one to another can be 2 methods:

1. Easy Transition
2. Transition Map

The easy transition makes your action transit to another directly automatically.

Except Easy Transition, there is another transition method we can apply, it is Transition Map. Transition Map is able to precisely control how one action transit to another, which others action will in between of them. For example, the current action is “walk” and the target action is “stand”. If we are using Easy Transition, the action transition will be “walk -> stand” directly. Whereas if we are using Transition Map, the result will be “walk -> walk to stand -> stand”, which guild by transition map. Please look at the picture below.

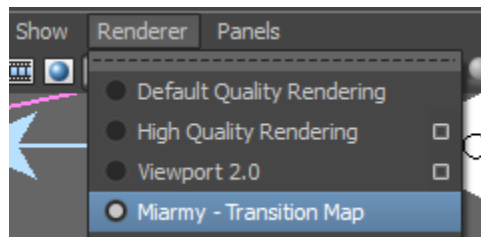
Also, the Transition Map can control action blending, action randomization, and then get the feedback from agent playback.



Transition Map screenshot

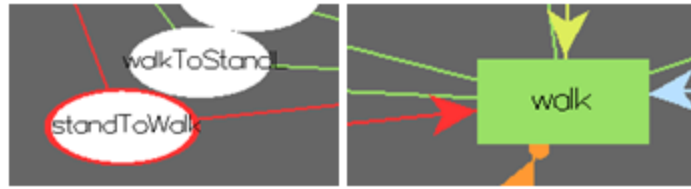
Note: Transition Map is always showing the contents of active agent. Please make sure you already activate the agent type which you want to edit.

Transition Map located in viewport renderer:



You should switch it to front view for viewing it:

Transition Map consists of “State”, “ActonShell” and “connections”



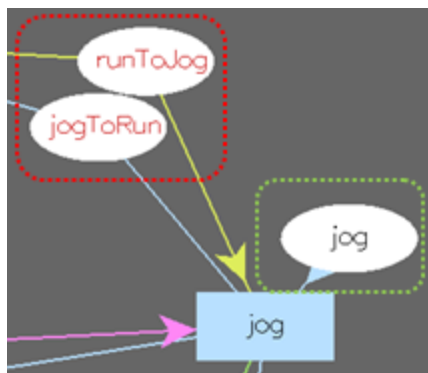
(left:) action shell (right:) state

- **Action Shells** are the aliases of actions. They stand for actions but not actions.
- **States** are terminals can hook many cyclic actions.

Main Features of Transition Map

Naming checking

The action and action shell are twins. Engine parses “action shells” in Transition Map whereas drives agents by “actions”. So both “action shells” and “actions” should exist with right naming conventions.



- The “runToJog” and “jogToRun” only have action shell without real action, the title of action shell will be “red”
- The “jog” has both action shell and action and can be recognized by engine. The title will be normal “black”

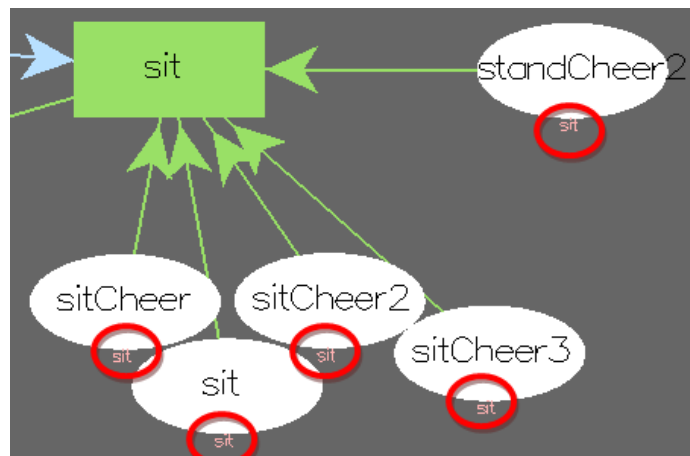
Precise transition action control

Between each 2 cyclic action, we can apply some transition actions between 2 states. To trigger the action, simply put your action name in decision node. All the things are automatically done by engine. The engine can always find the shortest way to get your target action.



Action group

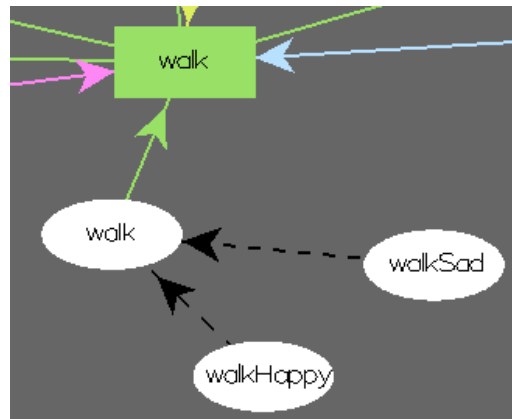
A group of cyclic action can make a “signal” group, when our logic decision call “actionGroup:sit” channel the system will automatically random select one of the actions of it.



Action blending

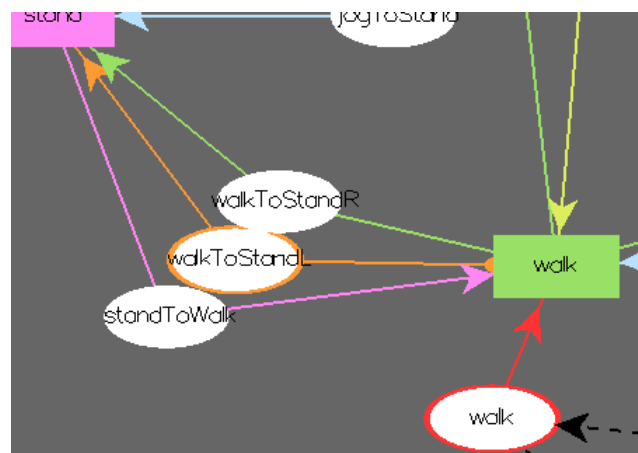
One action can blend to another action with blend engine. Not only the action itself, the rate can be blended. Blend is from 0% to 100%. 2 blend actions should have the same number of frames!

Call “walk->walkHappy” in decision node can perform this:



Exits choice

Between 2 cyclic actions



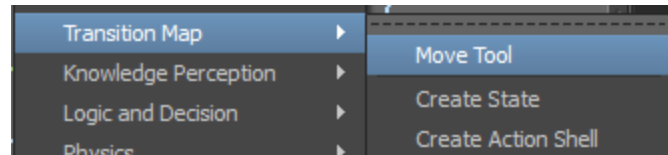
Take a look at the above picture, from walk to stand, there are 2 ways: “walkToStandR” and “walkToStandL”, we want to choose one of them in different situation. More specifically, different phase of action should transit to appropriate transition action.

Exit Choices:					
Exit Action:	walkToStandL	Start Frame:	5.0000	End Frame:	10.0000
		Preview:	0	exit	27
Exit Action:	walkToStandR	Start Frame:	18.0000	End Frame:	22.0000
		Preview:	0	exit	27

We just need specify the Exist Choices in form of Action Editor, everything will be ok. You may notice the right blue-white bar shows the exit phase.

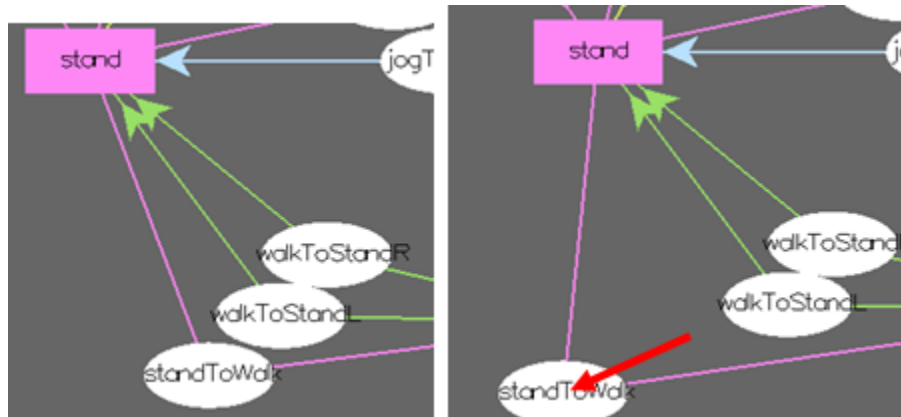
How to Build Transition Map

In Transition Map renderer, switch to “Front” View:



Enable Move Tool firstly, and there are 3 types of operations in Transition Map,

1. click-hold for moving nodes



Move the nodes

2. Above pictures, ctrl + click "walkHappy", and ctrl + click walk, for connecting



Link 2 nodes

3. Above pictures, shift + click walkHappy, and shift + click walk, for disconnecting

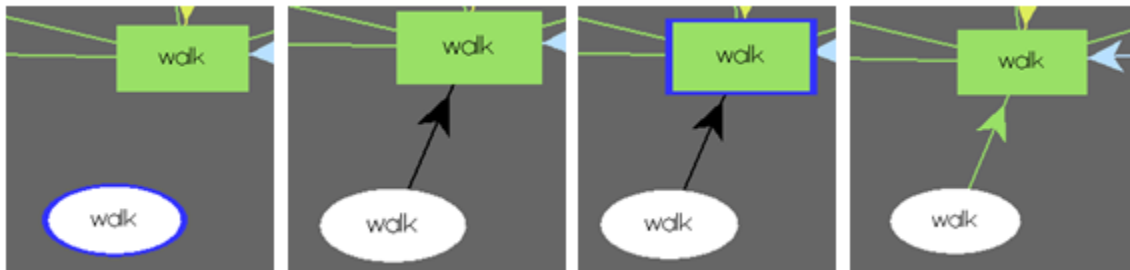


Break connections

And there are 3 types of Action Connections in Transition Map (enable move tool firstly):

1. The cycle action:

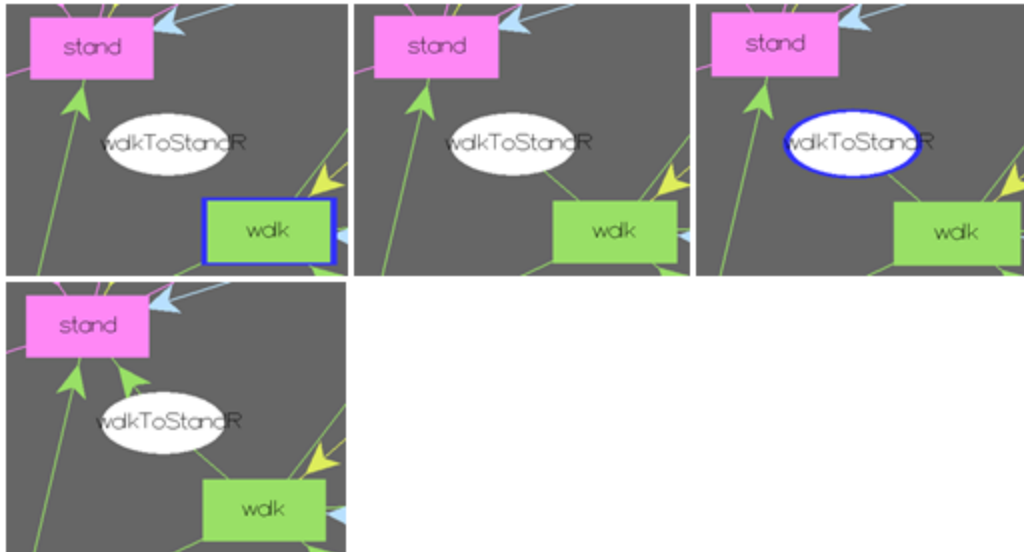
- a. ctrl + click walk action
- b. then ctrl + click walk state
- c. ctrl + click walk state
- d. ctrl + click walk action



Cycle action host on a state

2. The transition action:

- a. Ctrl + click walk
- b. Ctrl + click walkToStand
- c. Ctrl + click walkToStand
- d. Ctrl + click stand



Transition action between 2 states

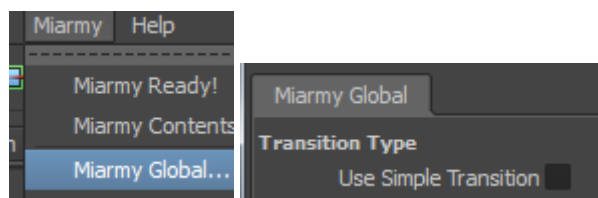
3. Action Blend:

- Ctrl + click walkHappy
- Ctrl + click walk

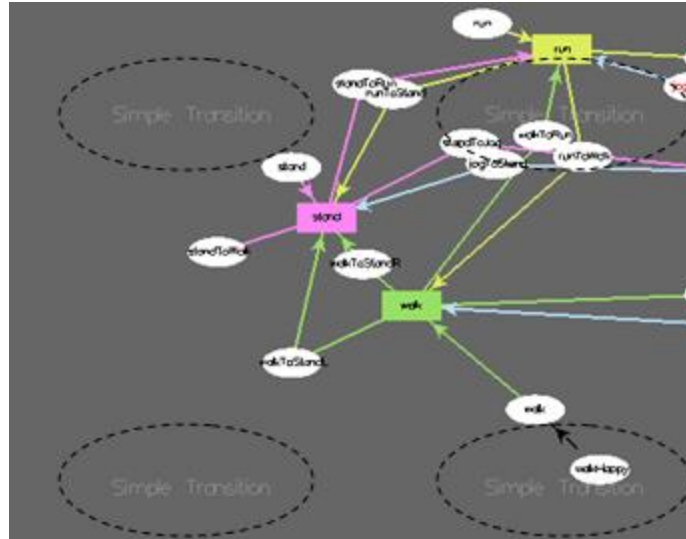


Blend action link to an another action

Uncheck the Simple Transition in Miarmy Global:



If you unintentionally enable Simple Transition, the Transition Map will have a background with tip message "Simple Transition" appear



Watermarks there in simple transition mode

Decision

Decision node is another infrastructure of agents' brain. Please check out the details in **Part 7 Logic in-depth** and **Part 8 Logic & Perception**

Create Decision node

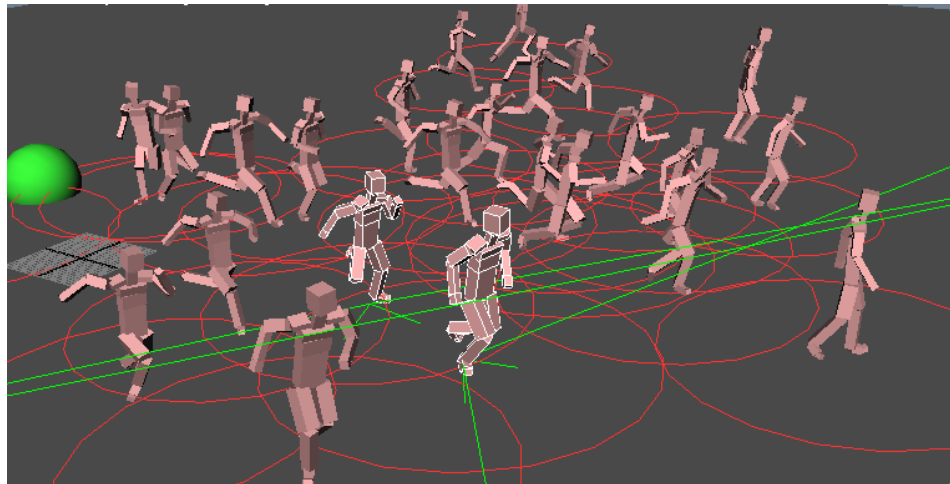
Firstly, you need active the agent type which you want to edit.

You can easily create it from Miarmy > Logic Decision > Make Decision

Agent

There is much information on each instantiated agent. Some of them are exist in the form of attributes, located in channel box, whereas some of them in memory and cannot access by regular methods.

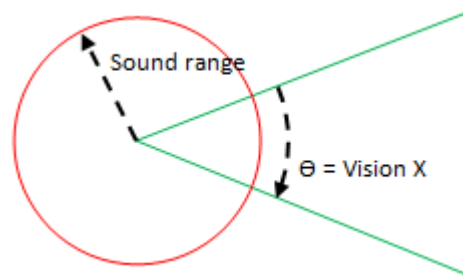
Notice: the information (attributes and memory data) on agent cannot be saved when you are saving your scene. All of the information is originated from **Original Agents**. In this version of Miarmy, Only "muteDynamic" attribute can be stored in place node through inverse placement feature



Acting agents

Agent attributes

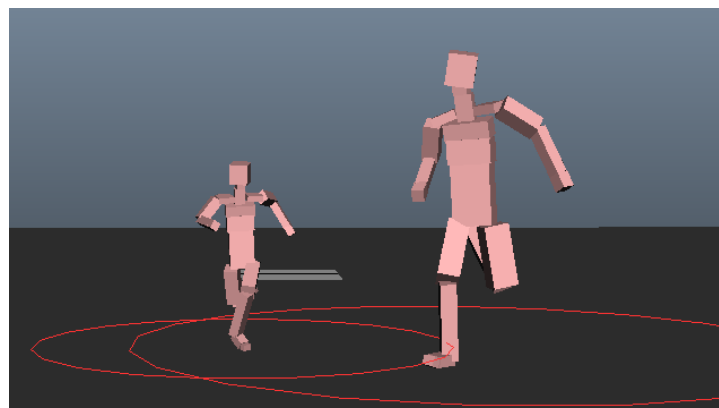
```
Mute Dynamic off  
Agent Scale 1.038  
Sound Range 34.859  
Sound Freq 1  
Vision X 100  
Vision Y 50  
Color Index 0
```



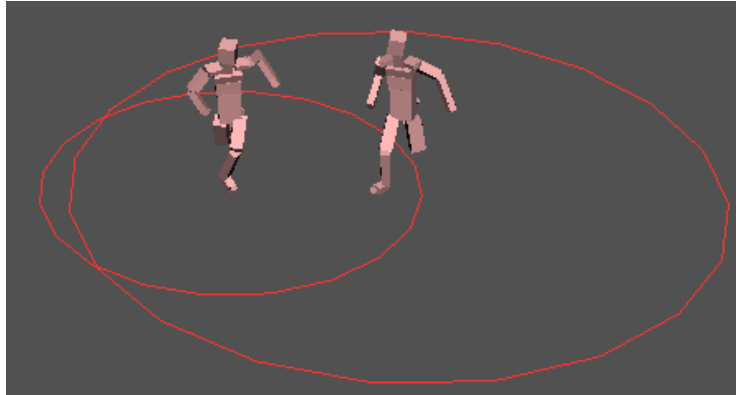
Mute Dynamic. Once enable, this agent can entirely block the dynamics feature, and this attribute can be stored in place node through inverse placement.

This is a very important feature for optimizing crowd dynamics, and we will introduce it in detail in **Part 9 Optimization – Mute Dynamics**

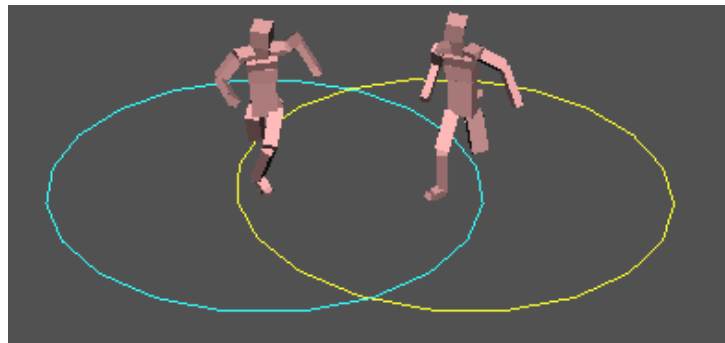
The others attributes, originated from Original Agent



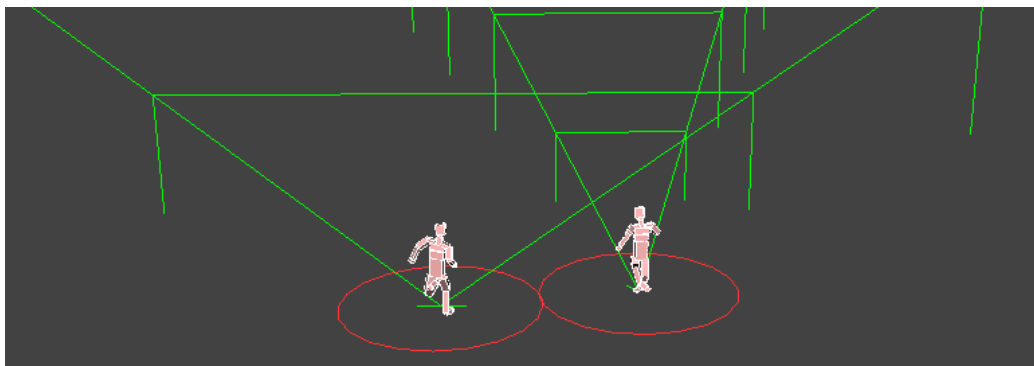
Agent scale 0.8(left) and 1.3(right)



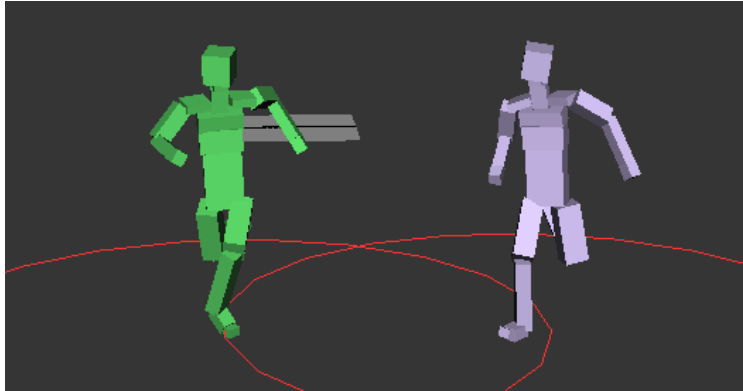
Sound range 35(left) and 60(right)



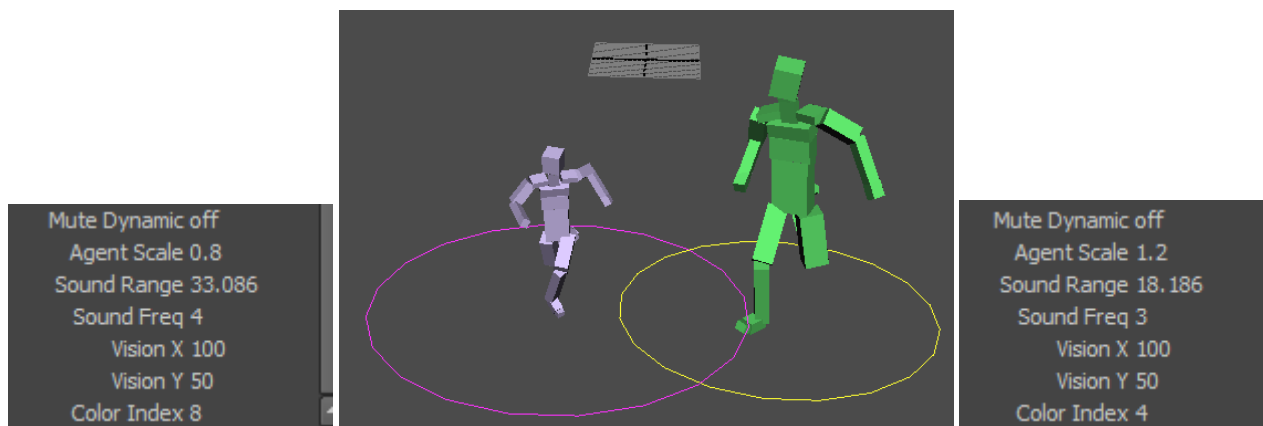
Sound frequency 5(left) and 3(right)



Vision X 100(left) and 30(right)



Color 4(left) and 8(right)



Compare the left and right attributes differences and agent results

Agent Mode

There are 3 modes of agent:

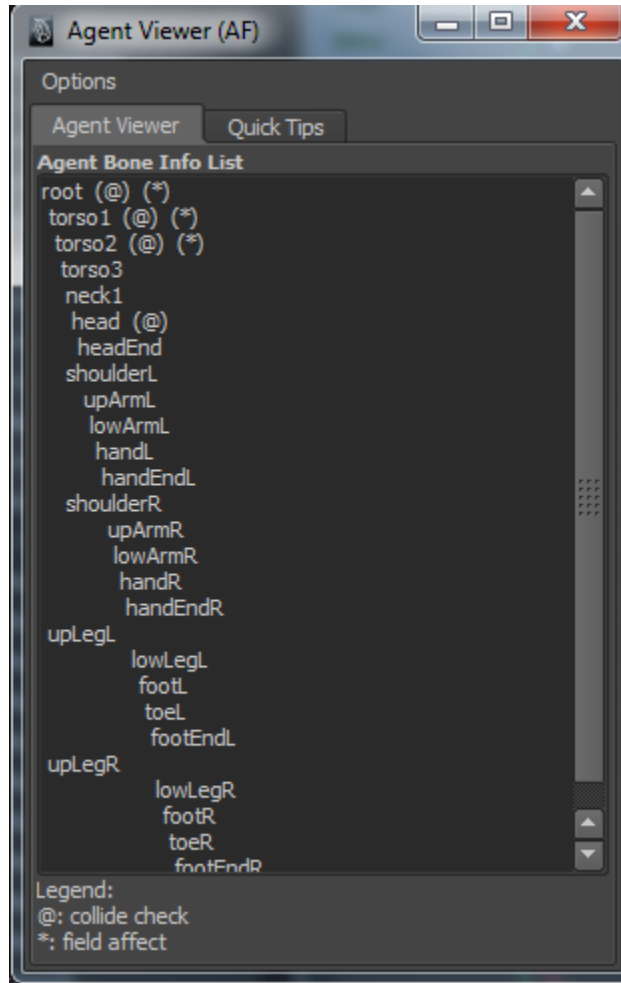
- **Acting mode:** controlled by action and logic decision.
- **Dynamics mode:** controlled by action and logic decision.
- **Combo mode:** some parts controlled by action whereas another parts controlled by dynamics, such as these feature enable: breakable dynamics, partial dynamics and body dynamics

Agent Memory Data

You can check the bone structure and flags on bone through Agent Viewer. Because these data cannot be saved, so we didn't provide any method to modify them.

Take a look at the following picture, the tree like list is the bone structure. The bone name is the "real name" in memory and if you want to fill the bone-specific channel in logic decision node (such as bone "upArmR:tx"), please use the name in this list.

The bone maybe has some flags, such as the **root**, have “@” and “*” flags. Check the legend on the bottom of the list, the “@” stand for the collision detection flag, and the “*” stand for field affect flag. We will talk about the collision detection channel in **Part 5 Logic & Perception**, and the field dynamics in **Part 7 Physics**



Agent viewer, contains bone structure and flags

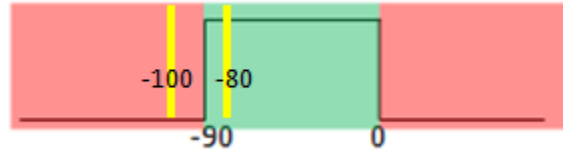
Part 4 Logic in-depth

This part explained all the logic engine calculation and procedure. All of them are happened under the hood automatically and it will not need you understand all of them at this time. You can combine with the video tutorials and make everything clear gradually.

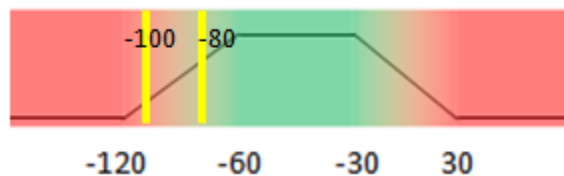
Fuzzy Logic

In the world of fuzzy logic, there is no “True” or “False”, because the boundary of “True” and “False” have been blurred. Instead, we called them “active”. If true, it’s 1.0 active, and if false, it’s 0.0 active, between true and false, it’s a float point value from 0.0 to 1.0 active, such 0.618 active.

Make an example and take a look at the pictures below, in regular logic, an input value is -80, the output should be “**True**”, and another input value is -100, the output should be “**False**”. But in fuzzy logic, the input values -80 and -100 are neither true nor false. Notice the slope in in the second picture, the input -80 get **active** value 0.667, and the -100 get the **active** value 0.333



Regular Logic: **Red**: False, **Green**: True



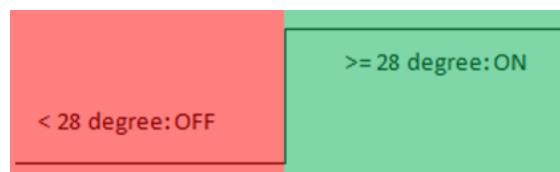
Fuzzy Logic: **Red**: 1.0 Active; **Green**: 0.0 Active; **Gradient**, 0.x Active

So our Miarmy system is full Fuzzy logic system implementation. And all the calculations are based on “**Fuzzy Active**” rather than “**True/False**”, including sentence test, priority and node active test.

Look at an example:

This is regular logic we are talking about, something true, do 1, something false, do 2:

1. When hot, turn on fan.
2. When cool (not hot), turn off it.

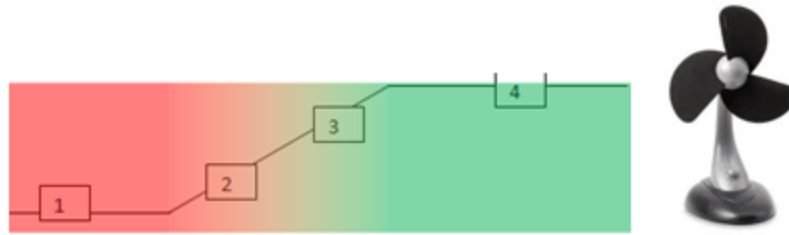


Color means fan speed, the dark line is the temperature

Here we bring in a concept called “**active**”, ranging from **0 to 1**, please take a look at the picture below

- When **false**, the **active** is 0
- When **true**, the **active** is 1
- The active value 0 and 1, represent the true and false

Then, this it is fuzzy logic, it make the boundary blurred:



Color means fan speed, the dark line is the temperature

1. When cold turn off fan (the **active** value is 0.0)
2. When a little bit hot, turn on fan (the **active** value is 0.2)
3. When hotter, turn up fan (the **active** value is 0.8)
4. Very hot, turn up fan speed to max (the **active** value is 1.0)

Pipeline

Terminology

As we mentioned in the Part 1 Main Concept, in a single decision node, there are several conditions and decisions. In terminology, we are going to call them **sentence inputs** and **decision outputs**.

There is an **input channel** in each sentence input whereas an **output channel** in each decision output.

Please distinguish the “**active**” and “**results**”, the active is a fuzzy logic concept value and it would be and usually in range from 0.0 to 1.0, whereas, the results can be any values, it’s a variables. E.g. channel “tz” active is 0.55, channel “tz” value is 100, and the results is “ $100 * 0.55 = 55$ ”

The calculation result of an input channel is an array, and it would be arbitrary number of results (even 0-length array and no result), we called them **channel results**

The calculated active of a single sentence, we call it **sentence active**

The calculated of a decision node, we call it **node active**

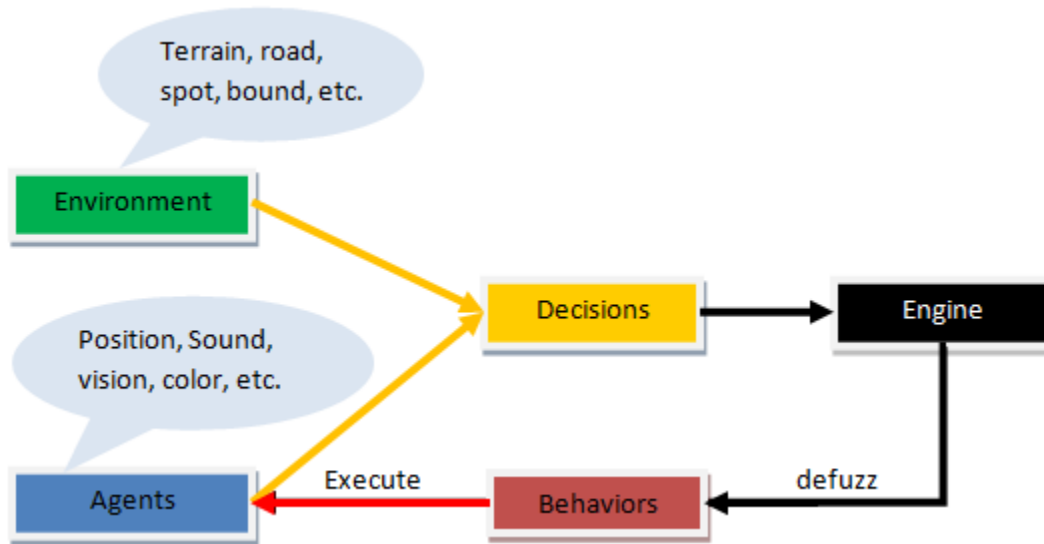
After node active, we will get some output channel results, we call them **decision results**

We need solve out the **behavior results** from decision results, and this calculation process we call it **defuzz**

General Pipeline

Our brain node will collect all the information from scene (environment perceptions & agents themselves) and put the information to Miarmy Core Engine. Then our engine will deal with the information and solve out the behavior results, finally drive agents and update the scene in brain post node.

The engine is not a node or command, it's a piece of code can receive information, solve out the results and send information back



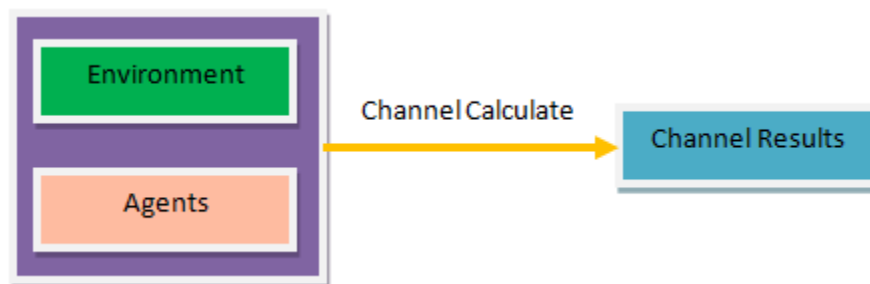
Logic General Pipeline

Engine Process Breakdown

1. Calculate each sentence input **channel results**
2. Solve out the **sentence active** from the sentence channel results by fuzzy logic
3. Use sentence active and logic rule and priority among these sentences, solve out the **node active**
4. Using hierarchical priority ranking system, recalculate the **node active**
5. Collect all the **decision results** based on node active
6. Using defuzz engine and decision results to calculate **behavior result**

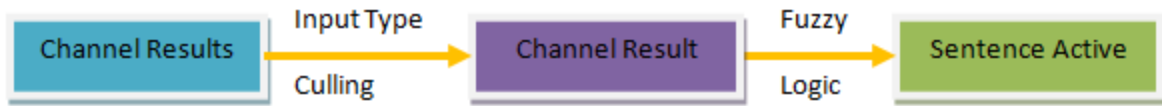
Please notice the color of the following pictures:

Firstly, we need solve out the channel results from the type of input channel



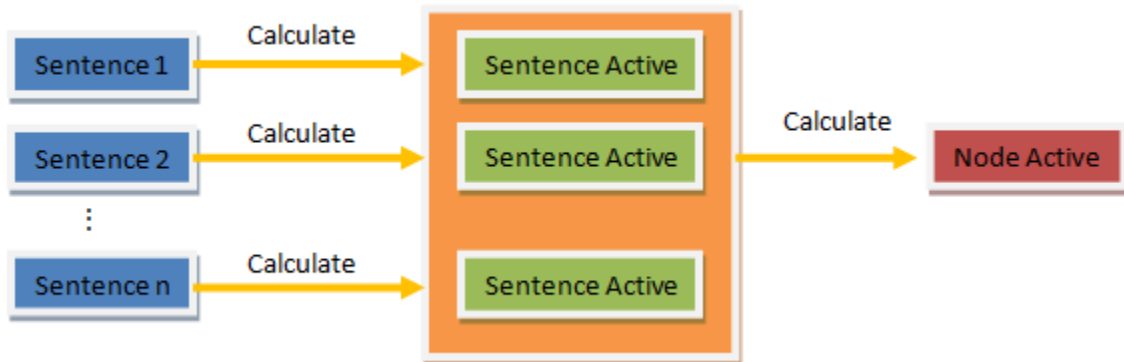
Step 1: from input channel, calculate the channel results

After getting the values from input channel, we will use the test range, fuzzy value and input type to calculating out the active of a single sentence, we called this sentence active



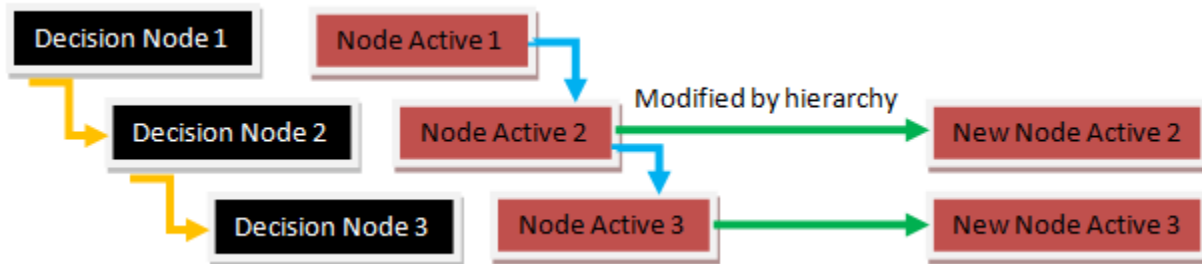
Step 2: Sentence Active Calculation Pipeline

Then we need use the **logic tool** and **priority** among several sentences and to calculate out the **node active**



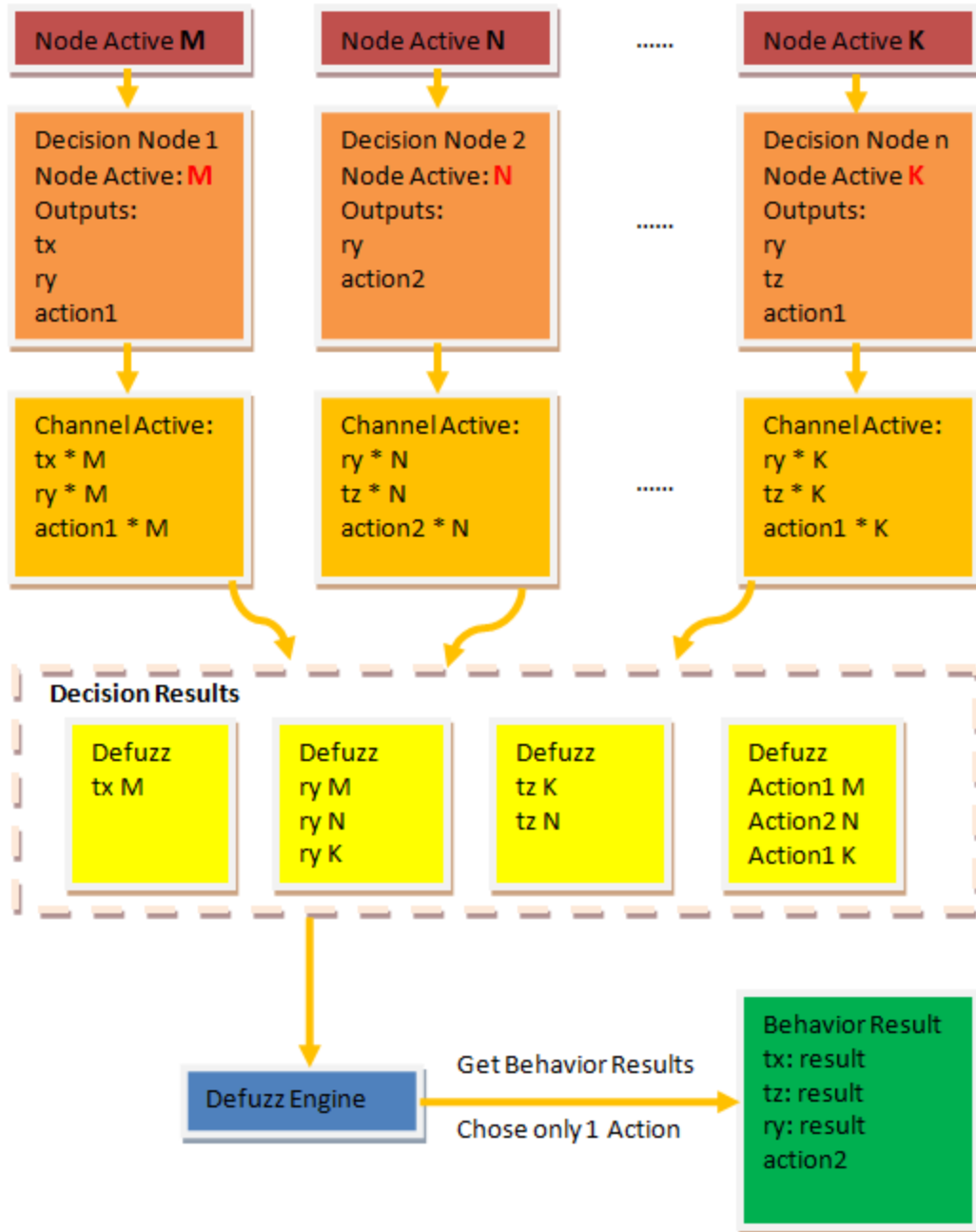
Step 3: Node Activate Calculation Pipeline

Using priority ranking hierarchy, recalculate out the new node active



Step 4: Modify node active using priority ranking hierarchy

Using the active value from each node, and decision output values, we can get many **decision results**. Then using defuzz method, we can solve out the actual results of each type of channel, drive behavior

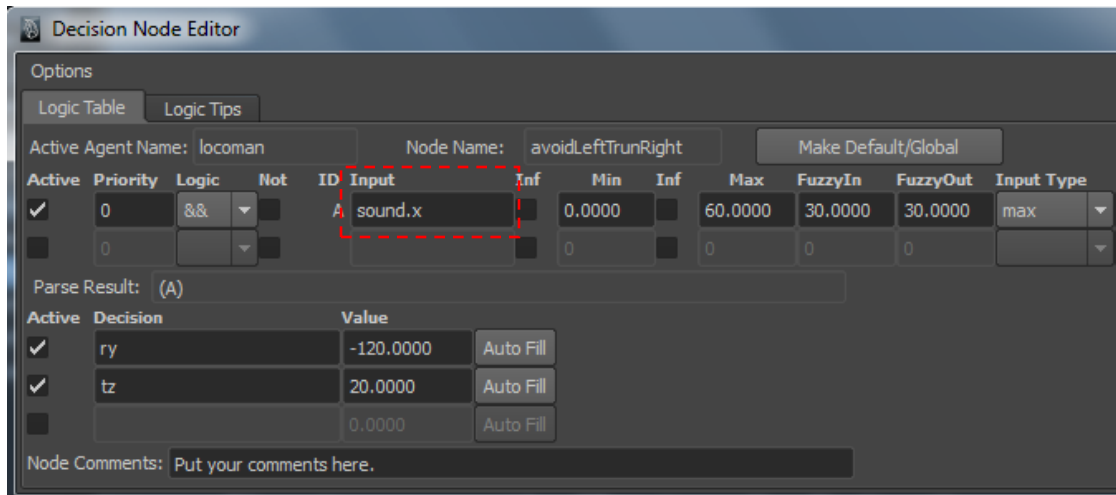


Step 5 & 6: using node active and decision results finally get the behavior results, defuzz

Implementation

Step 1: Get Channel Results

From input channel name, we can let the engine calculate the results for this channel, the result is an array, and the length of array is non-fixed, even can be zero.

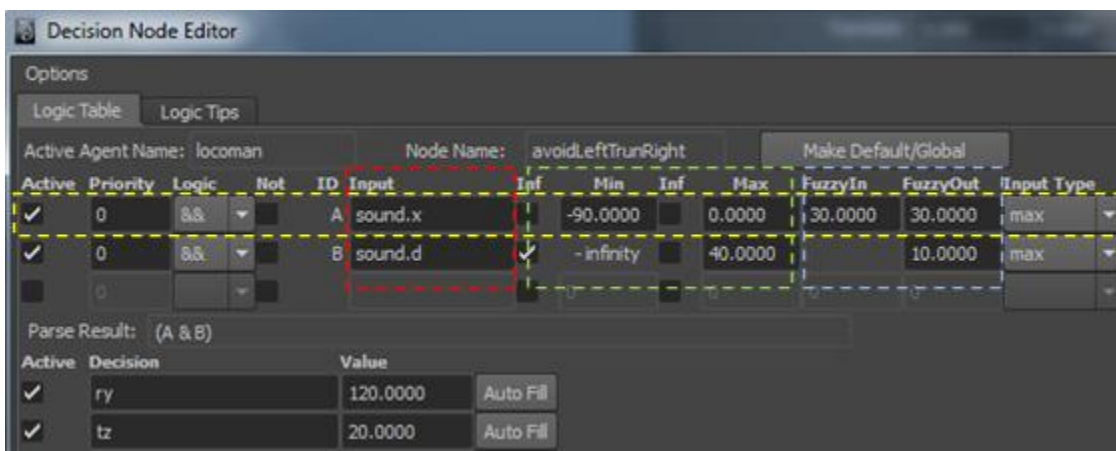


A sentence of node, Red: input channel

The channel “sound.x” may return many results based on how many agents in the sound range of current agent. Also maybe it returns nothing. (See “sound” perception chapter for getting the details the return values of “sound.x” channel)

Step 2: Fuzzy Logic for a Single Sentence Calculation

In the following picture, a single sentence contains an input channel, a true range and 2 fuzzy values.



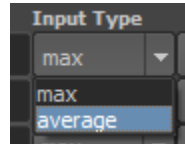
A single channel in yellow bound, Red: input channel Green Bound: true range Blue: fuzzy values

Each sentence is composed of:

- **Input channel:** return a list of values based on channel type
- **Test rule:** ID, pre-operator, priority, inverse flag
- **Test range:** the full true range without fuzzy blur
- **Fuzzy value** for the start and end of test range, plus and minus

Firstly, the input channel will return a list of values from Channel Input Engine base on the information of agent itself and environment.

Using input type, we can deal with the list of input results.



The input type have “max” and “average” 2 kinds of calculation algorithm:

- Max: choose the value make the sentence most be activated **one**
- Average: take all the results from that array and calculate the average **one**

Finally, we can get one result value, we call it **Channel Result**. Please distinguish it with the **Channel Results**

Use the Channel Result being input value and use fuzzy logic to test its output active, this active is sentence active

Fuzzy Range

And the sentence contains a “true range”:

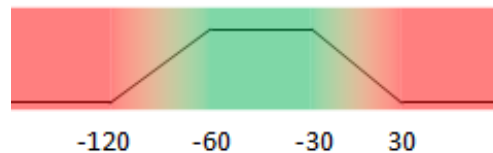


True range from -90 to 0 make sentence true

Then and a fuzzy value, such as 30, then turns out the fuzzy range:

=> $(-90 - 30)$, $(-90 + 30)$, $(0 - 30)$, $(0 + 30)$

=> -120, -60, -30, 30



Fuzzy range from -120 to 30 make sentence activated

Then the sentence active value will be:

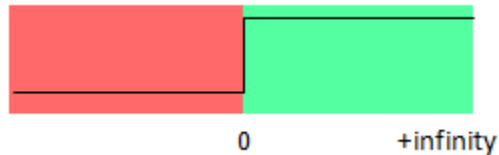
- Full active 1.0: if **Channel Result** from -60 to -30
- Non active 0.0: if **Channel Result** less than -120 or greater than 30
- Float point active 0.x: if **Channel Result** from -120 to -60 or -30 to 30

If using “Inf” channel (infinity channel), enable, and the fuzzy out will disappear

Inf	Min	Inf	Max	FuzzyIn	FuzzyOut
<input type="checkbox"/>	0.0000	<input checked="" type="checkbox"/>	+ infinity	20.0000	

0 to + infinity in channel

The true range will be like this:

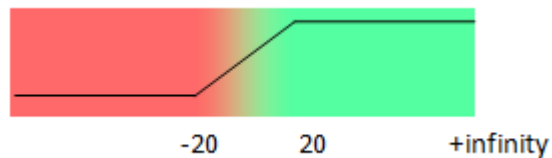


True range: input > 0 true

Then and a fuzzy value, such as 20, then turns out the fuzzy range:

=> $0 + 20 = 20$; $0 - 20 = -20$

=> -20, 20



Fuzzy range: input greater than -20 make sentence activated

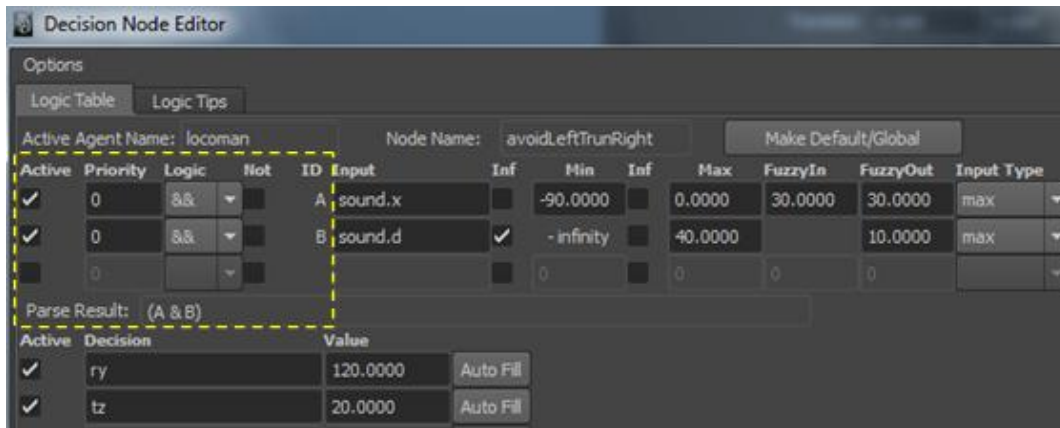
Then the sentence active value will be:

- Full active 1.0: if **Channel Result** greater than 20
- Non active 0.0: if **Channel Result** less than -20
- Float point active 0.x: if **Channel Result** from -20 to 20

Testing that **Channel Result** with the fuzzy range, we can get the “**sentence active**” in this step

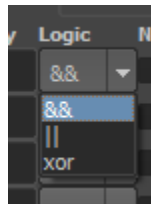
Step 3: Fuzzy Logic between Sentences Calculation

Among the different sentences, there are “active flags”, “sentence priority”, “not’ flags”, “logic operators” and the ID of each sentence. Using these logic tools and the “sentence active” of each sentence from previous step, we can calculate out the “node active”



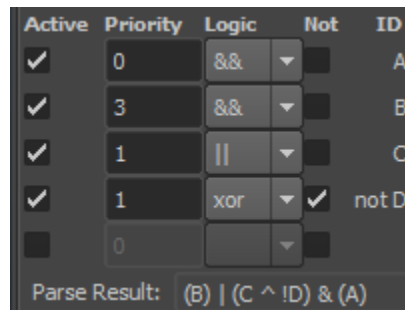
Yellow bound are logic tools

We can give a logic operator before the each sentence



&&: “and” operator, ||: “or” operator, xor: “exclusive or” operator

You can also using the “not flags”, “sentence priority” modify the result, and the calculation rule will be conclusion in **Parse Result**, use this rule and sentence active from each one, we can finally get the **node active**



An example of more complicated situation, but usually we don't need that

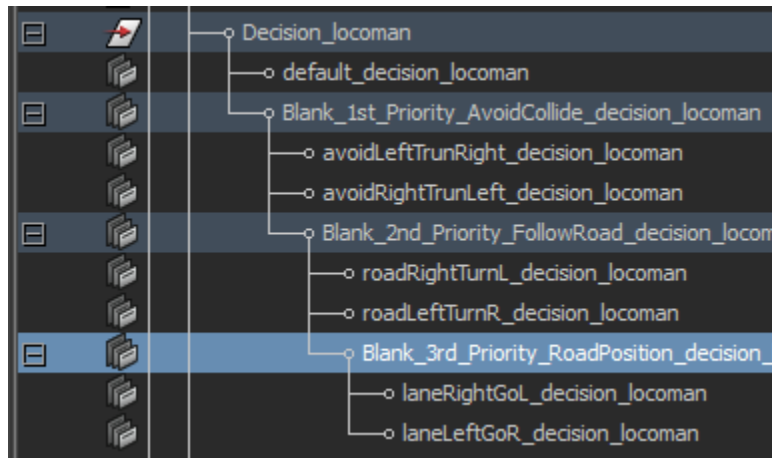
Algorithm for prefix operator:

- Add: the minimums active value
- Or: the maximums active value
- Exclusive Or: the absolute value of the different

After the calculation among these sentences, we will get the node active, in this step.

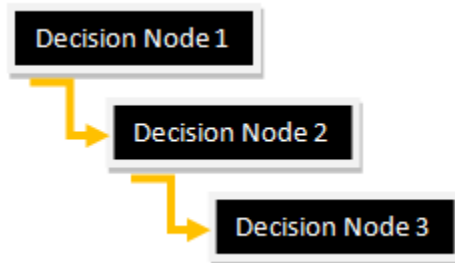
Step 4: Fuzzy Logic for Priority Ranking and Interfere to Node Active

By now, we have already known the **node active**, every node has its own node active, but if they are re-arranging by hierarchy, the node active will be interfered. The decision in higher hierarchy takes higher priority.



A real example of decision hierarchy

Direct hierarchy:



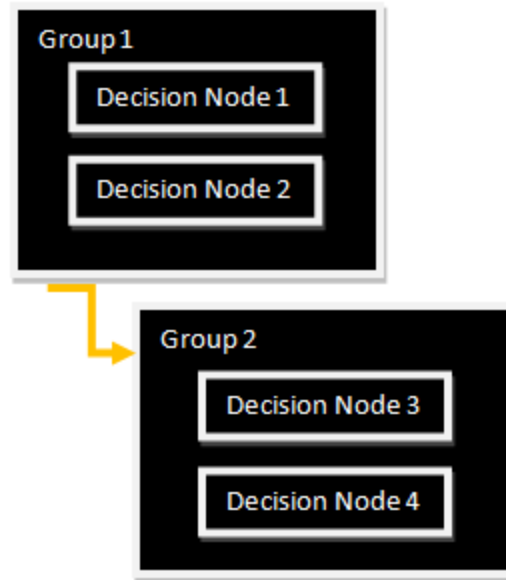
(1 – Node Active) will be passed to the child, and multiple to the Node Active of child

For example, the node 1, node 2, node 3 active values are: 0.2, 1.0, and 0.5

- the node1 active is 0.2, then the $0.8(1-0.2)$ will be passed to node2
- the node2 active is 1.0 and it will yield to $0.8(1.0 * 0.8)$, and the $0.2(1.0 - 0.8)$ will be passed to node3
- the node3 active is $0.1(0.5 * 0.2)$

So the node active results of the 3 nodes are: 0.2, 0.8, and 0.1 after ranking priority.

Grouping hierarchy:



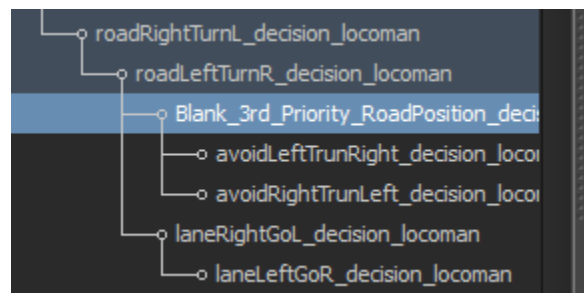
$(1 - \text{Max Active})$ will be passed to the child group, and multiply to all of the children decisions

For example the node active values from node1 to node 4 are 0.1, 0.4, 0.7 and 0.5

- the max active value of node1 and 2 is 0.4, so $0.6(1.0 - 0.4)$ will be passed to group2 and multiple to node3 and node4
- so the post-calculated active value of node3 is $0.42(0.7 * 0.6)$ and node4 is $0.3(0.5 * 0.6)$

Grouping and Node Hierarchy Hybrid

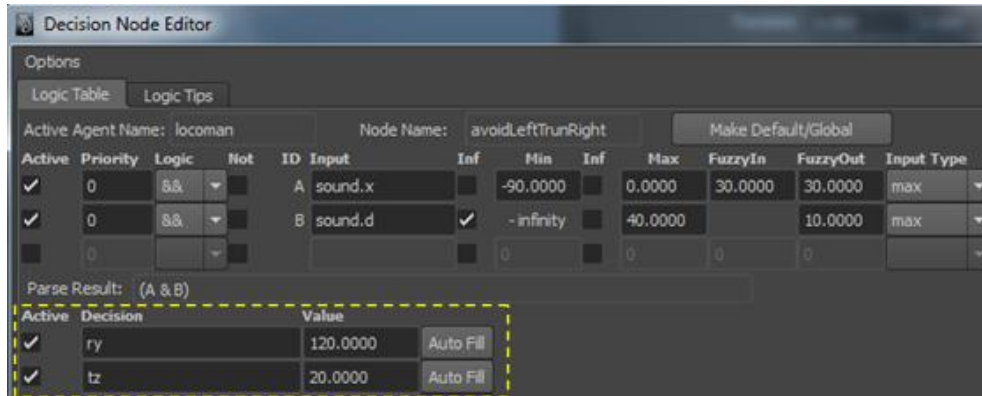
You also can combine the direct and grouping method together, and the algorithm is the same



Hybrid hierarchy ranking priority

Step 5 & 6: Output Mechanism - Node Active, Default Decision and Normal Decision

Using each node active value, we can firstly get the output channel active of each node, and then solve out the decision results. Just like the picture below, if the node active is 0.2 after sentences calculation. The “ry” and “tz” channels are active 0.2 for this specific node. The decision results are “ry”: 24 ($120 * 0.2$) and “tz” 4 ($20 * 0.2$).



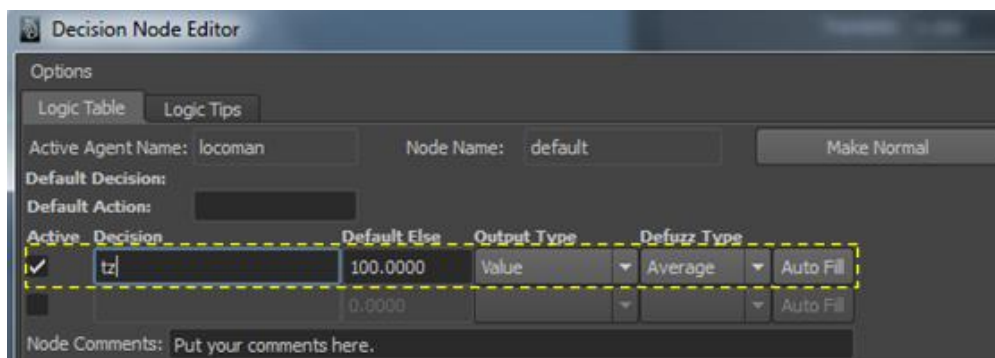
Normal Decision determine output channel based on node active

But please notice, this output channel active (like the 0.2 we just mentioned) is not the final output active for this type of channel, system need also calculate the node active from every node. And finally, system need solve out the ultimate correct active value of a type of channel combine with several active values from every node and the default value of this channel in the default decision.

Before continuing, we need now introduce a concept called default decision.

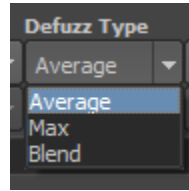
- Default decision implicitly contains the default value of every channel's output. (even there is no default decision, these implicit values exist)
- The default value means the value which output channel will blend with, when this type of output channel has not been fully activated by any of normal node.
- One can explicitly specify one or more default values for some specific channels and those default values will take place the implicit ones.

If we want to get the ultimate active value of specific channel, we need combine with default decision node.



Default value will work when this channel is not fully activated

Please look at the picture above, “tz” channel has been explicitly re-specified and when there is no any node trigger “tz”, the “tz” channel will be 100.0 by default; And if the “tz” channel is fully triggered by any other node, the tz will take the value from that node; Or if the “tz” channel has not been fully triggered, system will calculate a blended value base on Defuzz Type Algorithm.

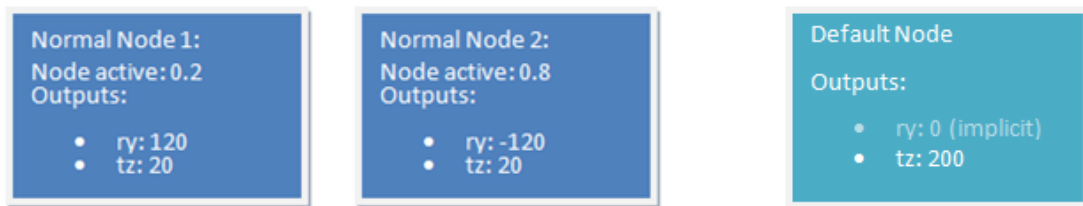


3 types of defuzz algorithm

- **Average:** average result from every normal decision, then average the default one
- **Max:** get the max one from result of normal decisions if the max one less than 0.5, get the default one
- **Blend:** don not distinguish the normal and default decision, blend the result all.

So far as we know, each node has a node active, and each node has several outputs. The part would a little bit confusion so we will explain this directly by an example. Imagine that we have bunch of decision nodes, and each one of them has several outputs. There must be several overlapped channel names. We need calculate the output active based on the defuzz type. Let’s take a look an example, just look at the following 3 nodes, 2 normal decisions and a default decision.

We just demonstrate the calculation of channel “tz”.



2 normal decision nodes with 0.2 and 0.8 active and a default node

For the first normal node, we got:

- value: 20
- active: 0.2

For the second normal node, we got:

- value: 20
- active 0.8

The default decision node, we got:

- Value: 200

We can calculate the result by following rules:

- **Average Type**
 - The default node active is $(1 - \max)$: $1 - 0.8 = 0.2$
 - $\text{Sum} = 0.2 * 20 + 0.8 * 20 + 0.2 * 200 = 60$
 - $\text{Sum of Active} = 0.2 + 0.8 + 0.2 = 1.2$
 - $\text{Result: } 60 / 1.2 = 50$
- **Max Type:**
 - We get the max active **0.8**, the second normal decision
 - Result: **20** itself (the second normal decision value)
 - Note: if the max active blew 0.5, we take the default value in default decision **200**
- **Blend Type:**
 - We get the max active **0.8**, the second normal decision
 - Result: $0.8 * 20 = 16$
 - Note: similar like Max Type but we need multiply the active value.

Part 5 Logic & Perception

As far as we know we can use input channel get an array which contains the input results. So different channel channels get the different array based on our **channel input engine**.

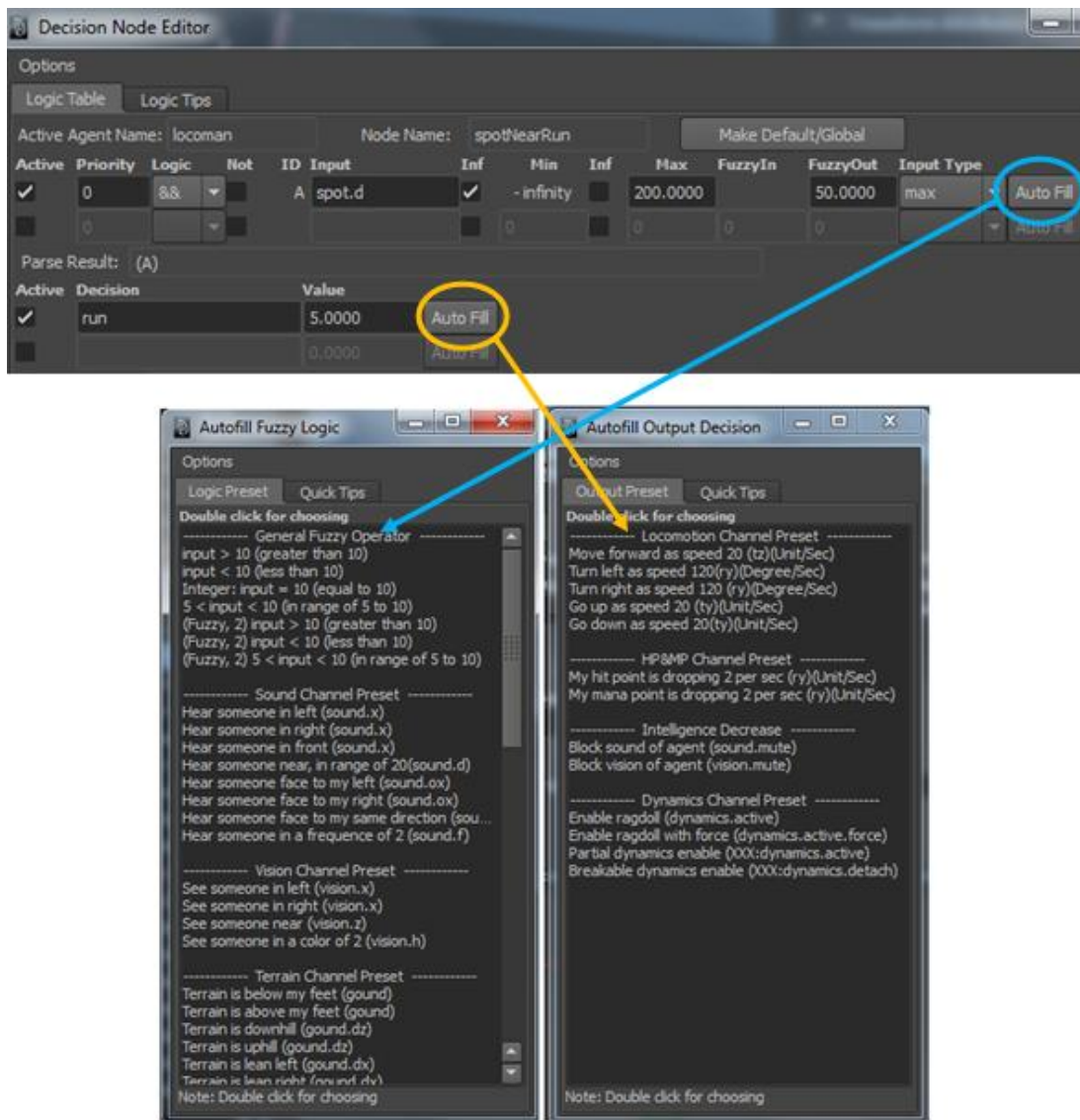
And also there are a lot of decision results represented by the output channel, we need transform that to the execute information by our **channel output engine** and send them to agent.

Logic Presets

There are many presets for building decisions or filling channels, after you familiar with all the concepts, you can use these preset accelerate the brain construction process.

Channel Presets

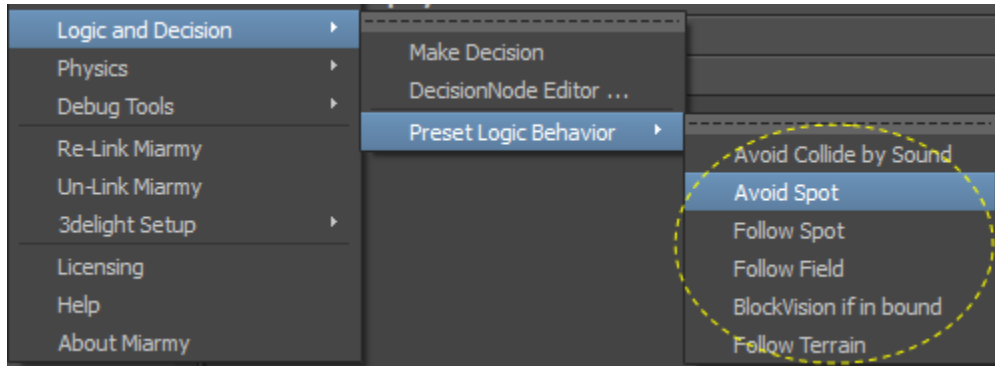
Automatically fill the channel contents



Channel presets

Decision Presets

Automatically create decision will logic inside.



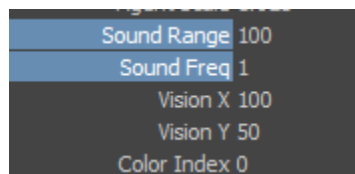
Logic decision presets

Interactive with Agent

Some channels can return result based on other agents, let the agent can feel others. They are “sound” and “vision”

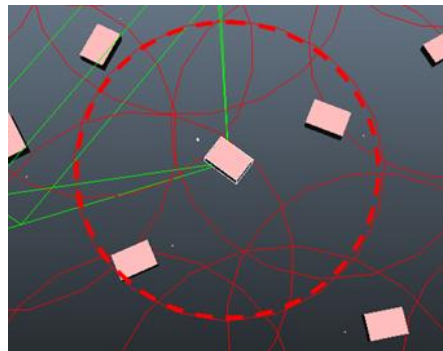
Sound

Sound channel is useful and effective ways for knowing the agent nearby and usually used to apply agents collision avoidance. Each of agents has an intrinsic sound range attribute and a sound frequency attribute. Selecting any agent, you may notice it is located channel box:



Sound attributes

Each of the agents has a sound itself. Like this:

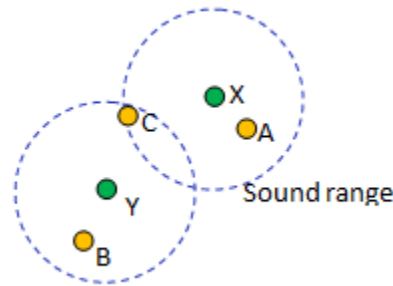


The sound range for agent

Sound channel have some auxiliary option, like `sound.x`, `sound.y`, `sound.d`, etc. No matter what channel you are using, before calculating, sound channel will firstly find the agents in its range, then only calculate result based the in-range agents. Like the picture shown blow, the Y agent can only feel B and C for sound testing, whereas the X will let agent C and A for testing.

The agent Y cannot feel agent A and X, and the agent X cannot feel agent B and Y

And actually the sound range is a 3D range, means it can feel the agent in top and button of it.



Select the agents in sound range firstly

Note: There's a concept call **subjective** and **objective**, like the picture shown above, when we testing agent Y, the "Agent Y" is subjective agent, we call it "**current agent**", and this time "Agent A, B, C and X" are objective agents, we call them "**others agents**". The engine will iterate calculate every agent in scene, and each one of them have one chance being the "current agent"

After getting the agents in the sound range of current agent, it will test them by specific channel and return values.

The `sound.x` will return the degree relative to the current agent. Look at the picture below,

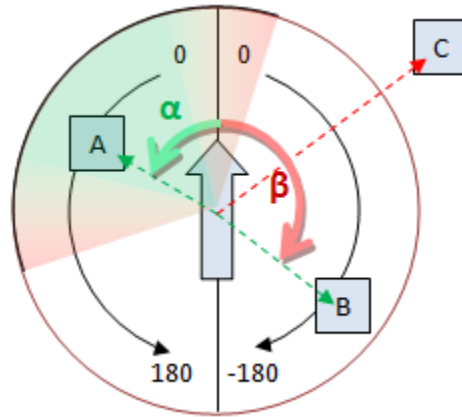
The `sound.x` will return 45 for agent "A", -135 for agent "B". As for agent "C", because the C agent is not in range of agent, so, there is no return. The final result will be an array which contents are [45, -135].

Using this array, we can combine the "input type and "fuzzy range", we can calculate out the sentence active.

Also see the picture below, for example, we are testing is there any agent in my left front, and the fuzzy range is 0 to 90, with a fuzzy value (the red gradient color). The results are [45, -135],

- So the 45 make the sentence fully active 1.0, and the -135 make the sentence active 0.0, if we using "input type: Max", we need find the maximum sentence active. Conclusion, the sentence active will be 1.0
- If we are using "input type: Average", we will take both sentence active 1.0 and 0.0 and average them, so the sentence active will be 0.5

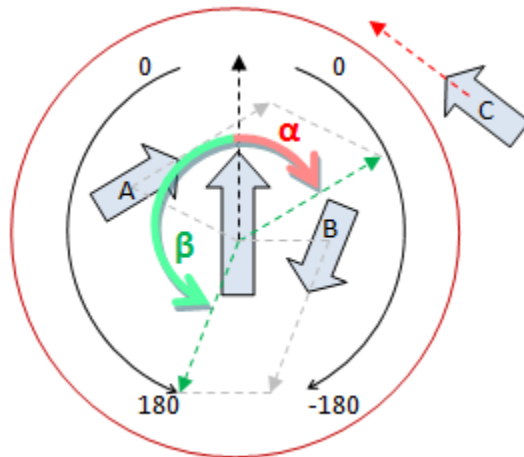
The `sound.x` is testing others agents in horizontal space whereas the the `sound.y` is testing others agents in vertical space



sound.x channel example for current agent

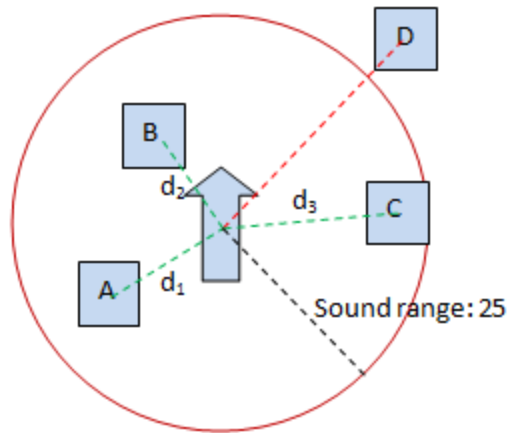
Take a look at the following picture, the `sound.ox` channel tests the in-range agent's orientation. The agent A, angle is α , should be -60 degree; for agent B, the angle is $\beta = -160$, in this example, the input results contents are [-60, 160]

The `sound.ox` is testing agents in horizontal space whereas the the `sound.oy` is testing agents in vertical space



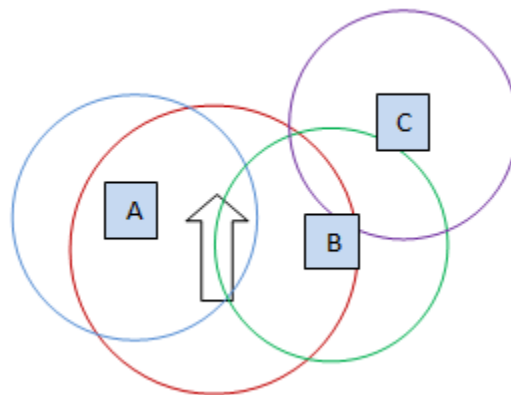
sound.ox channel example for current agent

`sound.d` channel will return the every distances between the current agent and the others agents in sound range. For the following example, only the agent A, B, C can join calculation. Assuming that the sound range of current agent if 25, the input results should be [d_1 , d_2 , d_3], approximate [18, 16, 23]



sound.d channel example for current agent

sound.f channel will return the each frequency of agent in current agent sound range. For the following example, the current agent can feel A and B, if the frequency of A is 3, the frequency of B is 5, the input results should be [3, 5]

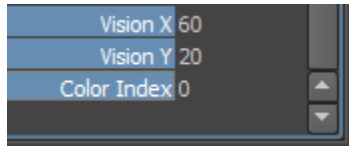


sound.f channel example for current agent

You may notice, with sound channel, we can easily get the information from nearby agents and response, so it's usually can use to aiming target, of avoid close agent.

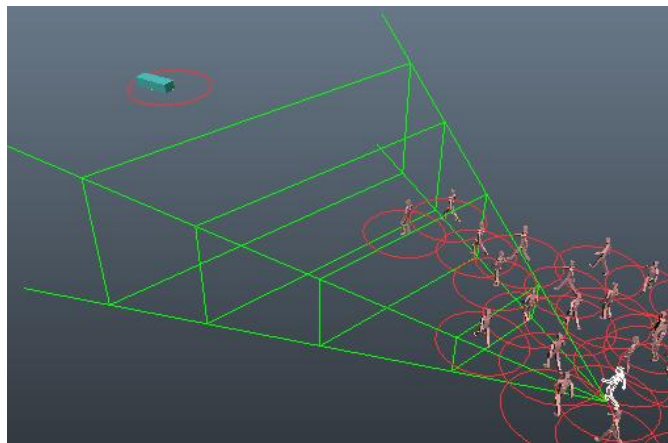
Vision

Vision channel is useful ways for finding identify far agent. Any agents in current agent vision range can be seen. Each of Miarmy agents has 2 intrinsic vision ranges attribute and a color attribute. Selecting any agent, you may notice it is located channel box.



Vision attributes

Each of the agents has a vision itself. Like this, notice the green frustum, it is the vision range of selected agent



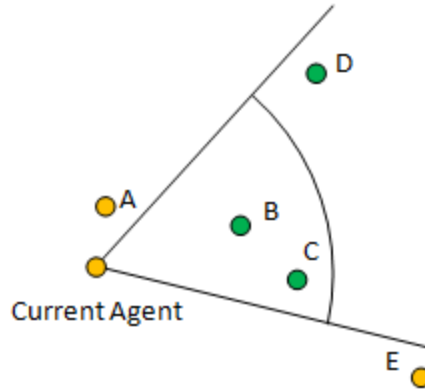
Vision range frustum

You can define these attribute in agent group node, see ***Part 3 Agent Infrastructure***

The same as sound, vision channel also have some auxiliary option, like vision.x, vision.y, vision.h, etc. No matter what channel you are using, before calculating, vision channel will firstly find the agents in its range, and then only calculate result based the in-range agents.

Like the picture shown blow, the current agent cannot see agent B, C and D, but cannot see Agent A and E

And actually the vision range is a 3D range as well as sound, means it can feel the agent in top and button of it.



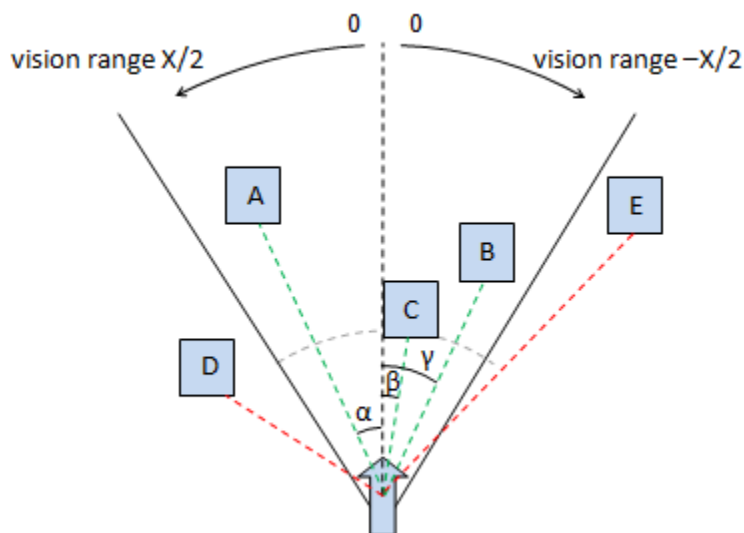
Select agents in vision range firstly

After getting the agents in the vision range current agent, it will test them by specific channel and return values.

The [vision.x](#) will return the degree relative to the current agent. Look at the picture below,

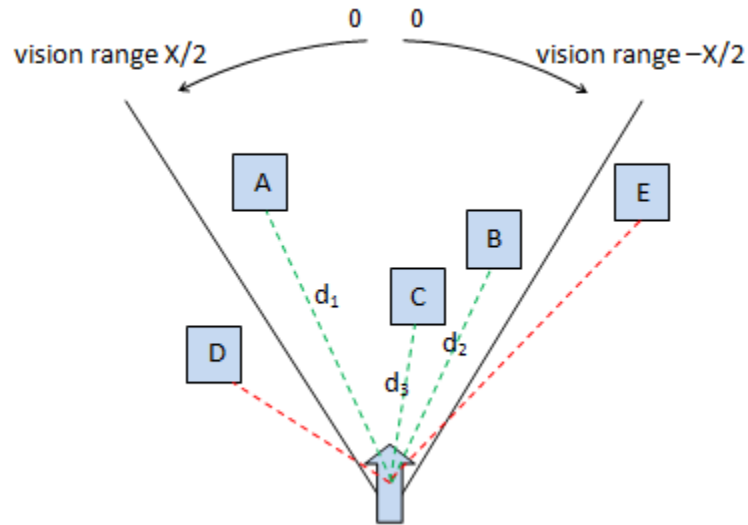
The vision.x will return α degree for agent "A", γ degree for agent "B", and β degree for agent "C". As for agent "D" and "E", because they are not in vision range of agent, so, there is no return. The final result will be an array which contents are $[\alpha, \beta, \gamma]$. As shown by following picture, the result approximate is $[30, 15, 35]$.

The vision.x is testing others agents in horizontal space whereas the the [vision.y](#) is testing others agents in vertical space



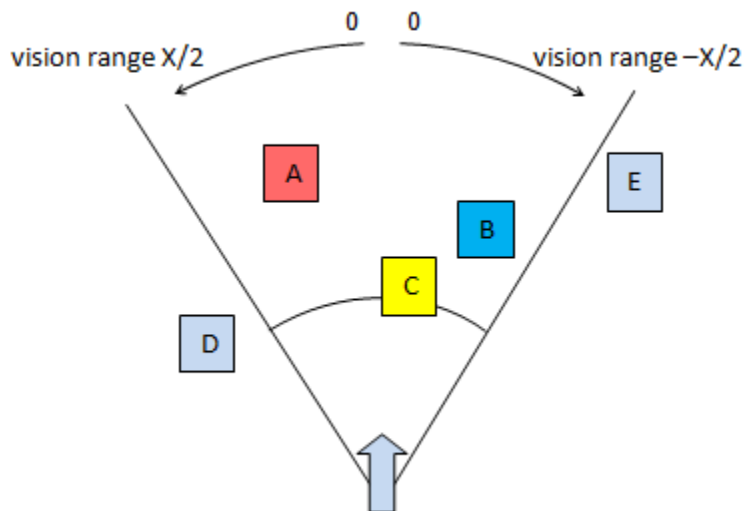
vision.x channel example for current agent

vision.z channel will return the every distances between the current agent and the others agents in vision range. For the following example, only the agent A, B, C can join calculation. Take a look at the example picture below, the input results should be $[d_1, d_2, d_3]$



vision.z channel example for current agent

vision.h (hue) channel will return the each color of agent in current agent vision range. For the following example, the current agent can see A, B and C, if the color of A is 0, the color of B is 2, the color of C is 3, the input results should be $[0, 2, 3]$



vision.h channel example for current agent

You may notice, with vision channel, we can easily get the information from far and in-front agents and response, so it's usually can use to identifying and chasing target.

Agent Exclusion Feature

Sound channel and vision channel has agent exclusion feature among sentence calculate, this feature can make sure the channel get more precise and natural results. Additionally, this feature is automatically enabled, and user cannot disable it.

Usually we need use not only one type of channel for interactive the others agents. For example, sometimes we need test is there someone in my left and near. (So that we can turn right and slow down)

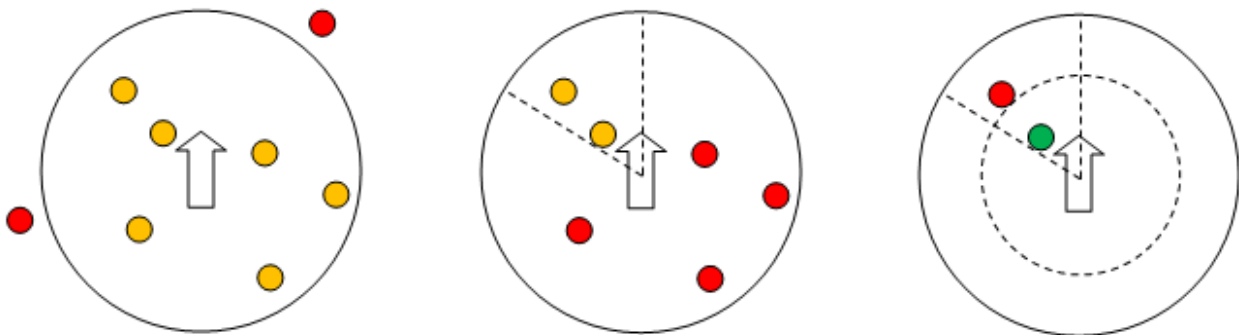
As far as we know, we can use sound.x channel to test is there someone in my left, and use sound.d channel to test is there someone nearby, then naturally use "AND" operator to get the intersection agents.

Just like this:

ty	Logic	Not	ID	Input	Inf	Min	Inf	Max	FuzzyIn	FuzzyOut
	&&	<input type="checkbox"/>	A	sound.x	<input type="checkbox"/>	0.0000	<input type="checkbox"/>	60.0000	30.0000	30.0000
	&&	<input type="checkbox"/>	B	sound.d	<input checked="" type="checkbox"/>	- infinity	<input type="checkbox"/>	15.0000		20.0000
		<input type="checkbox"/>			<input type="checkbox"/>	0	<input type="checkbox"/>	0	0	0
(A & B)										

Someone in my left and near

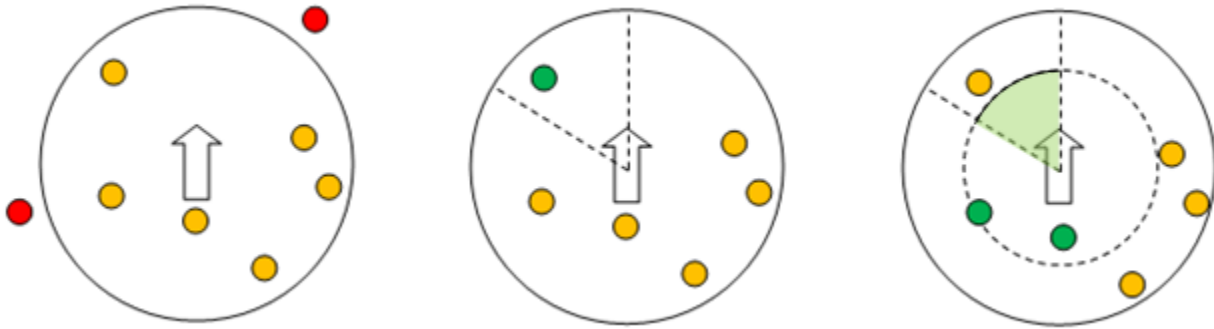
At this time, the agent exclusion feature will be automatically enabled. Please take look at the pictures below, the agents are gradually excluded by the filter and never join the calculation again for the other sentences in the same decision node, and we will finally get the accurate agent meet condition.



From left to right, the exclusion filters respectively are "sound range", "sound.x" and "sound.d"

Suppose that we didn't have Agent Exclusion Feature and take a look at the following example, the node will get the **wrong** result (wrong node active)!! Please notice the following second picture, green dot mean there is an agent make sound.x channel test active 1.0 and notice the third picture, green dots mean there are 2 agents

make sound.d channel active 1.0. Finally, sentence A and B will both active 1.0, the A “&&” B logic expression makes this node active 1.0. The system would mistake there is someone in my left and near. But unfortunately, in fact, there is no agent in the GREEN Zone in third picture! That is means that, actually there is no agent in my left and near. So, without agent exclusion feature, the channel will not very accurate.



Situation without agent exclusion feature

The vision channel also has agent exclusion feature enable. Therefore, between each sound and vision channel, we only need “&&” (“and” operator).

And also we need avoid sound and vision channels are filled in the same decision node, because the sound and vision channels have independent exclusive agent engine.

All of them happened automatically and the user will not notice it.

Dynamically Decrease Intelligence Mechanism (DDIM 1.0)

In some situation, for optimizing and speeding up of simulation, we need disable some feeling for some agents. And block the others agents feel them.

sound.mute (output channel)

If this channel is executed, the current agent will disable the sound and also cannot be hear by others agents.

vision.mute (output channel)

If this channel is executed, the current agent will disable the vision and also cannot be seen by others agents.

There is an example we explained more about this in **Part 9: Optimization - Lower the Intelligence**

Collision Detection

Collide channels are another important kind of feature make agent interactive each other. For details, please check out **Part 7 Collision Detection**

Interactive with Environment

One can create many Miarmy perception contents in scene and organize them to environment.

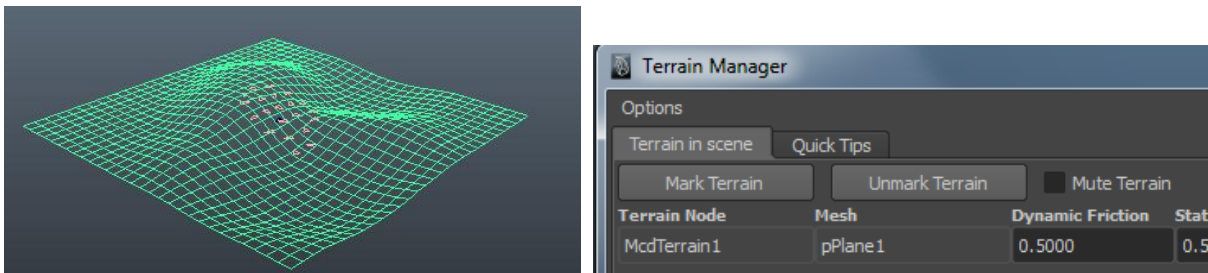
Some channels can return results based on environment, let the agent can feel the contents of the scene and interactive with them, like follow a road, avoid stuff and so on.

Ground

Ground channels let agents are able to feel terrains and interactive with them, adapt the shape of terrain, such like follow the height and align the orientation.

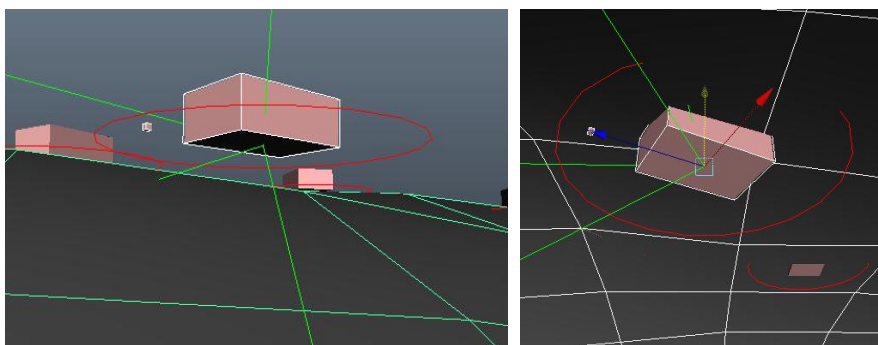
For making the agents are able to feel terrains, we need mark that piece of plane mesh terrain in Miarmy Terrain Manager. (We highly recommend that you simplify the mesh before marking it terrain)

And once a piece of geometry marked terrain, there is a McdTerrainNode connect to this mesh. This node will record some attributes of terrain for physical simulation usage. So you don't need do anything on this node.



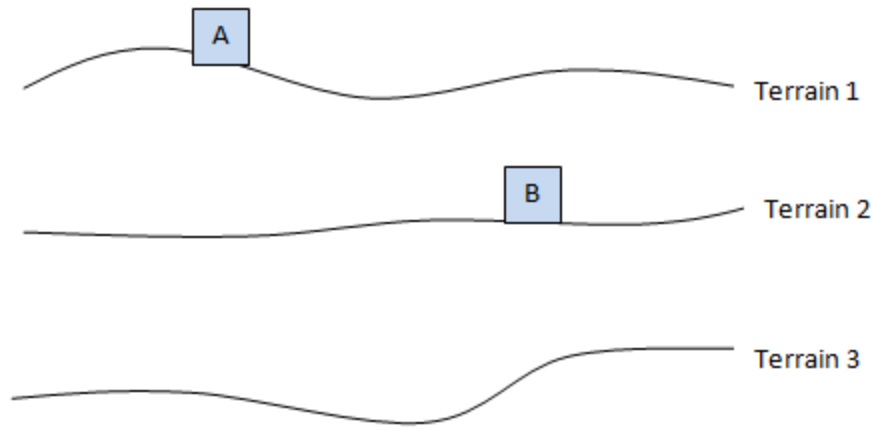
Terrain manager and mark terrain

Once marked terrain for a mesh plane, agent will find them when apply ground channels.



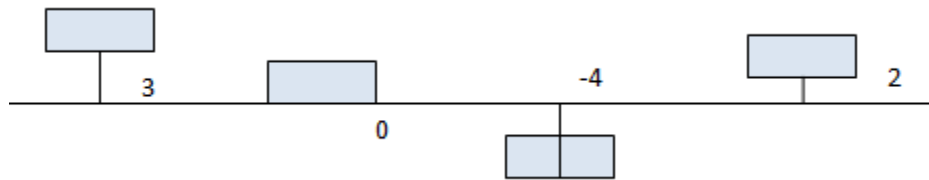
Above and beneath the terrain plane mesh

The ground channels will first find the closest terrain and return only one result to input results. Shown as the following picture, in this moment, the agent A can only feel and interactive with terrain 1 and the agent B can only feel and interactive with terrain 2



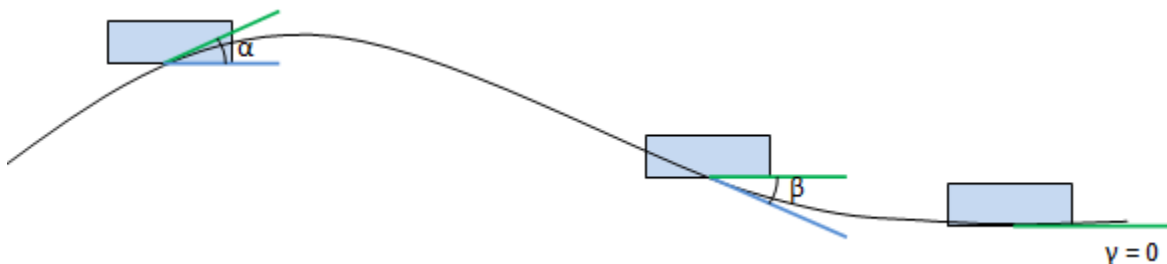
Agents are able to feel the closest terrain in scene

The **ground** channel will return the height underneath the feet of agent. For example and take a look at the following picture, the ground channel will return 3 for first agent, 0 for second agent, minus 4 for third agent and 2 for last agent.



The ground channel example for agents

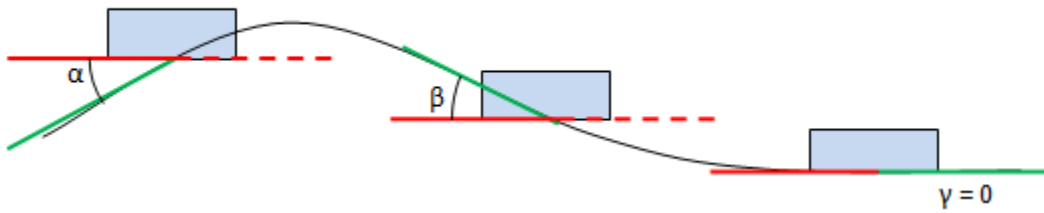
The **ground.dz** will return the angle between the Z-axis of agent and the terrain plane of that agent place, for example the following picture, the first agent will return $\alpha = 35$ because the ground in-front is higher, and the second agent will return $\beta = -40$ because the ground in-front is lower, whereas the third agent will return 0 because the ground in-front is align to Z-axis of current agent.



The ground.dz channel example for agents

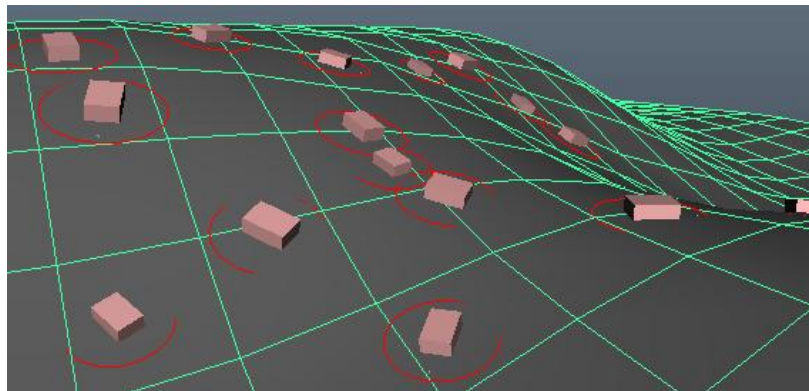
The **ground.dx** channel is very familiar with ground.dz channel, nothing special but testing the X-axis. In following picture, the first agent will return $\alpha = 40$ because the right terrain is higher, and the second agent will

return $\beta = -35$ because the left terrain is higher. The third agent will return 0 because the x-axis of agent is align to the terrain



The ground.dx channel example for agents

Once we get the relationship between the agents and terrain, we can move the agents up and down, rotate them forward or backward, and left or right for adapting the terrain



Using with ground, ground.dx and ground.dz, we can align our agent seamless to terrain

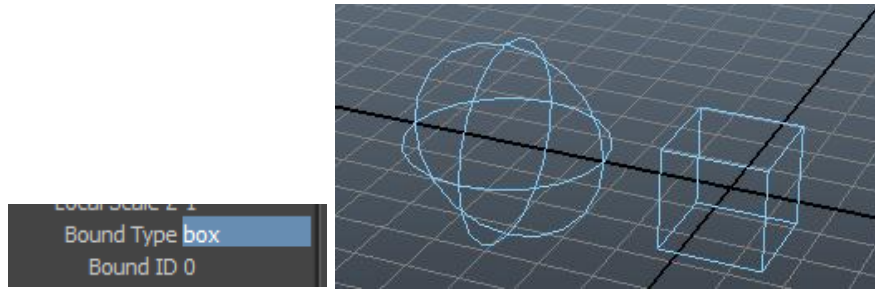
Note: the agents can read the information from deforming terrain, but it may not accurate when the deformation is big, because there is a Maya Bug. For avoiding the bug, we just need simply create a geometry cache for that terrain and remove the deformers.

Bound

Bound is a kind of perception contents, which can be cubic and spherical shapes in this version of Miarmy and you can key frame it for triggering some event easily, such as triggering a group of agent start to run. *(Bound can be any shapes in next version)*

You can create bound in Miarmy > Perception Contents > Create Bound, and each bound has 2 bound related attributes

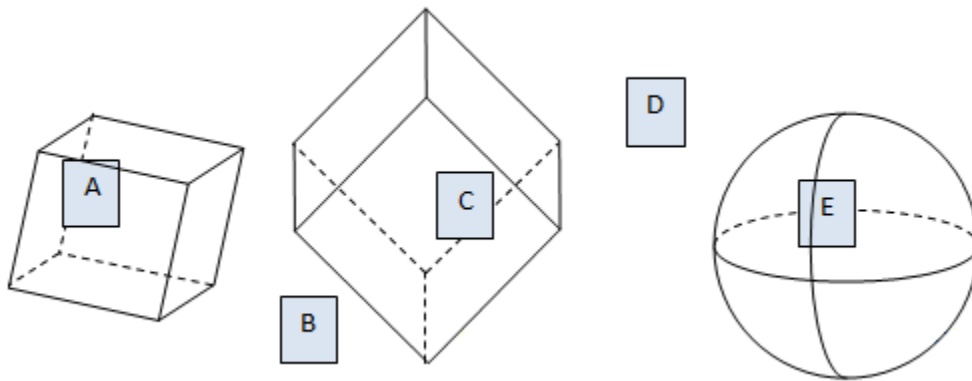
Bound channels let agents are able to feel bound and test whether in or out side of bound. It's the simplest channel but very useful.



(left:) Bound attributes (right:) Spherical and cubic bound

The **bound.in** channel will return 1 when the agent inside of it while 0 when the agent outside of it. Suppose there are 3 bounds in scene and 5 agents located in different places, so the inputs result will be:

- return 1 for agent A, C, E
- return 0 for agent B, D



Bound channel example

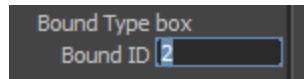
Indexing Technique

Some perception contents support indexing. The perception contents which can be indexed are:

- Bound
- Zone
- Road
- Wind

On each one of these perception contents, there usually is an attribute XXX ID, such as Bound ID or Road ID. The channel name can be modified to such as `bound[2].in`, `road[0].x` or `zone[15].d`. If the channel name have ID, it can only percept and interact with the perception contents with the same ID, for example, `road[3].ox` can only fetch results from the road which Road ID is 3.

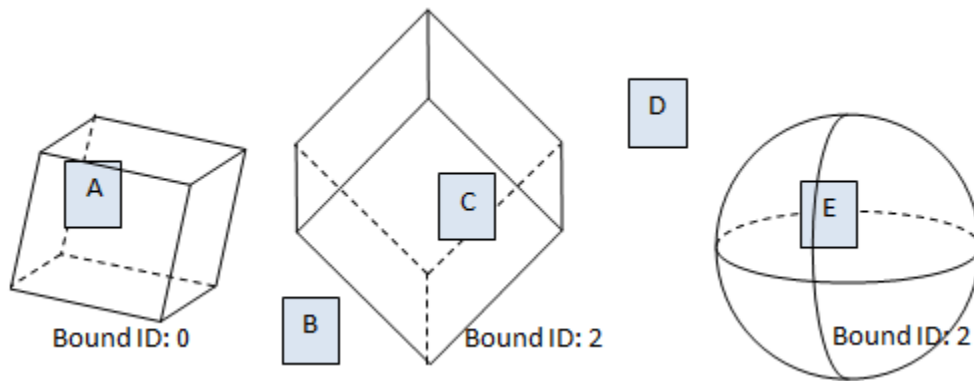
For example, bound is one kind of them. Please notice there is a Bound ID attribute on each bound.



Bound ID attribute can be any positive value (including 0)

We can simplify use `bound[0].in` to test whether current agent in bound which Bound ID is 0. So modified from the previous bound example, the results will be:

- Return 1 for agent A, because only A is inside of bound which ID is 0
- Return 0 for agent B, C, D and E

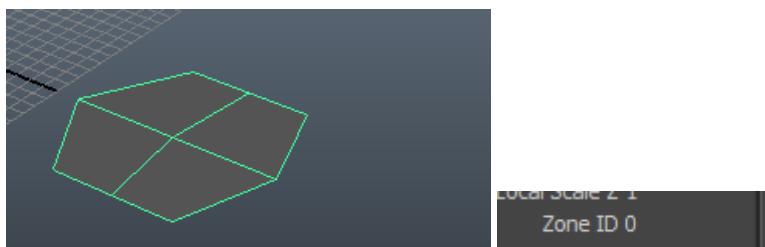


Bound channel with indexing techniques

Zone

Zone is a kind of perception contents created from a little piece of mesh, usually not complex one.

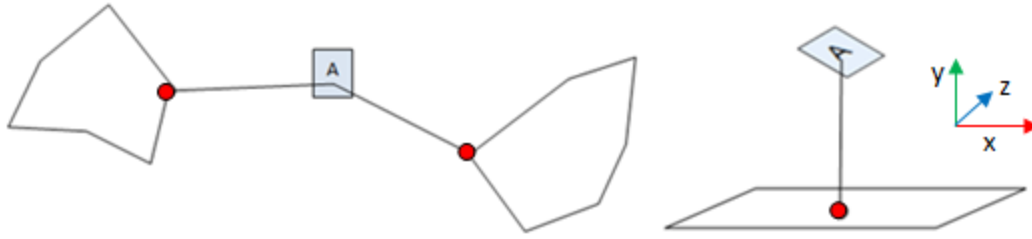
And similar like terrain, once a piece of geometry been marked zone, there is a `McdZoneNode` connect to this mesh. This node will record the Zone ID attribute for this zone.



(left:) any piece of simple mesh can be marked "zone" (right:) Zone ID attribute on zone

Zone channels let agents are able to feel a geometry zone plane with arbitrary shapes, and return the relationship to the zone.

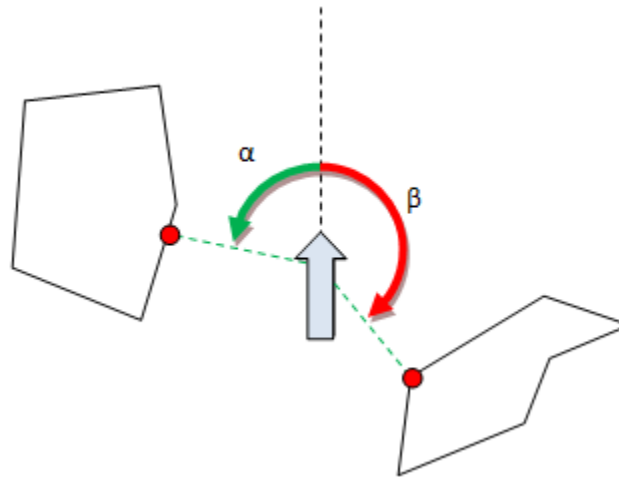
In fact the return results are based on the closest point on zone. So any zone channels will firstly find the closest points from zone, and any calculation is based on these points. The red dot on the edge of zone as shown below:



Find the closest point, (left) 2D space, (right) 3D space

The **zone.x** channel will return the degrees between the agent's Z-axis and the line from agent to the zones, in horizontal space. Look at the following picture below, the red dots are closest points and the angle α and β will be the returned input results, approximate $\alpha = 80$ and the $\beta = -140$, input results [80, -140]

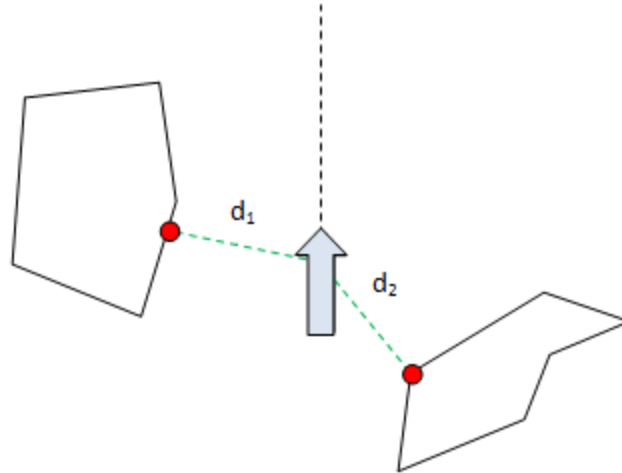
The zone.x is testing zones in horizontal space whereas the the **zone.y** is testing zones in vertical space



zone.x channel example

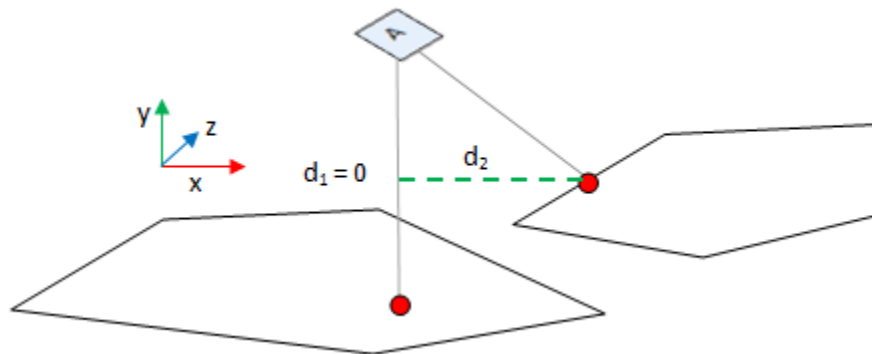
The **zone.d** (**zone.d2d**) channel will return the distances from agent to zones in XZ plane. Please take a look at the following 2 examples.

The first is in the top view, and the agent and 2 zones are all in XZ plane. After agent finding the closest points on plane, zone.d channel will return in distances in XZ plane, the input result will be $[d_1, d_2]$.



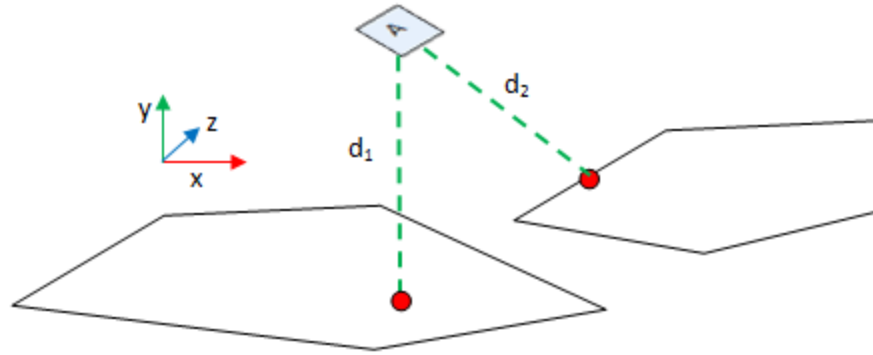
Example of zone.d or zone.d2d channel in top view

The second example shown this in 3d perspective view, please notice although the agent is in top of left zone but it is in side of it, the distance in XZ plane is actually 0. And the distance from the right zone should be mapped to the XZ plane d_2 instead of those grey lines.



Another example of zone.d or zone.d2d channel in perspective view

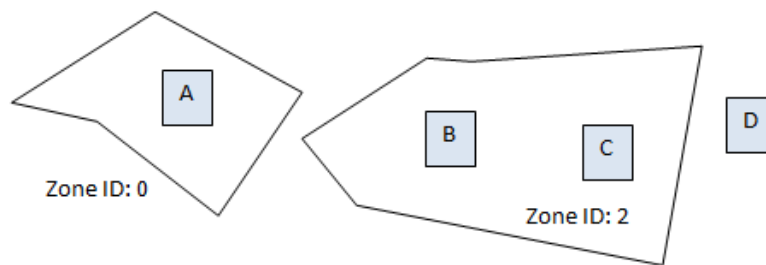
The zone.d3d channel will return distances in 3d space. Shown as the following picture, (please distinguish this to our previous zone.d2d example) we directly get the results from agent to the closest points of the zone. They are green lines in the following picture, without mapping them to XZ plane. The input results will be $[d_1, d_2]$



zone.d3d channel example in perspective view

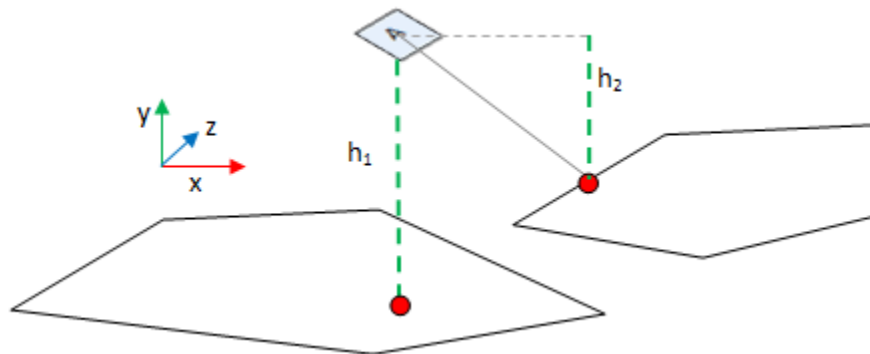
The [zone.in](#) channel can test whether current agent in the zone area in XZ plane and return 1 if the agent inside of it. Like the picture shown below, the agent A, B, and C is inside the zone area in XZ plane and will return 1 for zone.in channel. For agent D, zone.in channel will return 0 due to that it is not inside of any zone area.

If you using [zone\[2\].in](#) for testing them, the agent B and C will return 1 and agent A and D will return 0.



zone.in channel example with/without zone ID, in Top View

The [zone.hi](#) can test the heights from agent to the zones. Firstly we get the vectors from agent to the zones, and map those vectors to the Y-axis, like the green lines shown in the following picture, the channel input results should be $[h_1, h_2]$

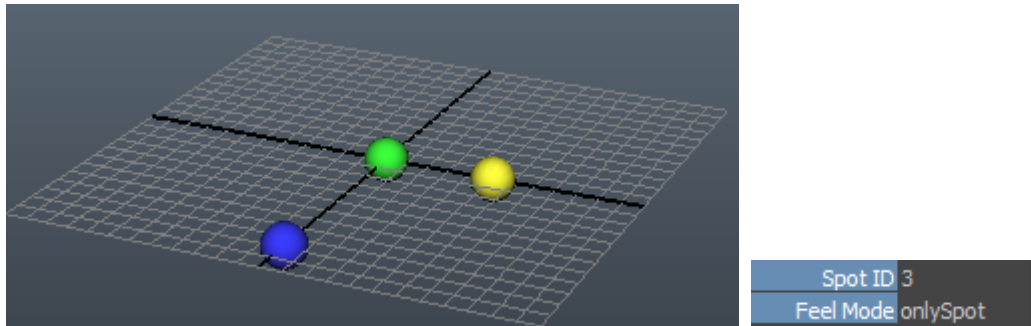


zone.hi channel example in perspective view

Spot

Spot is another perception content of Miarmy. It's actually a point in 3d space. With spot channels, agents can test distances and directions with the spots and interactive with them, such like follow a spot, or trigger some behaviors near a spot.

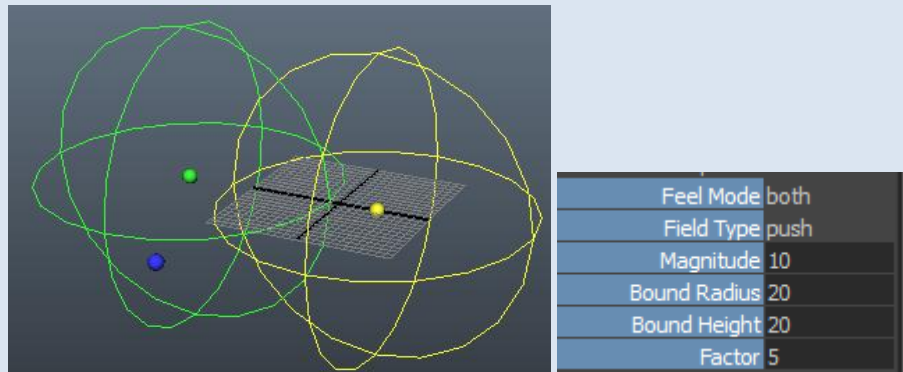
Additionally, spot support indexing.



(Left:) spots in scene, (right:) spot attributes

Note: you may notice there are some extra attributes on each spot, they are actually belongs to build-in force field, we designed spot with a build-in force field combo due to that in many case we need them both working together. We will talk about this later in

Part 7: Physical Simulation – build-in force field

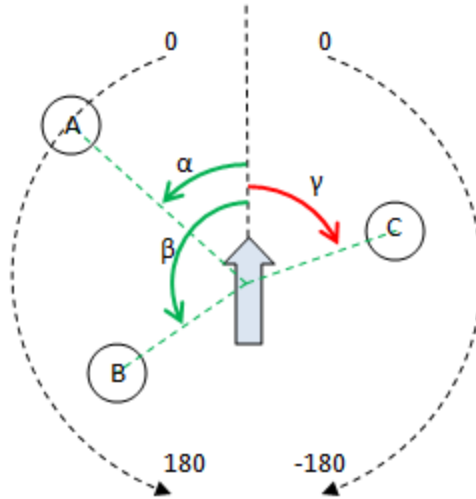


(left:) Spot and build-in force field combo, (right:)build-in force field attributes

You can create spot in Miarmy > Perception Contents > Create spot, and each spot has 1 spot related attribute.

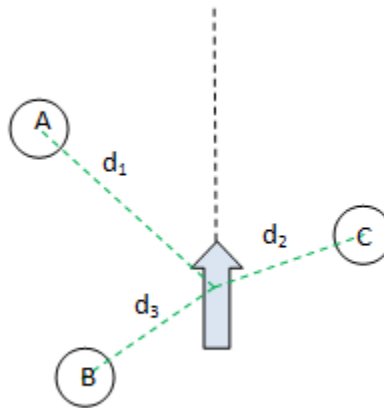
The **spot.x** channel will return the angle between the Z-axis of agent and the line of agent to the spots (angle between black line and green lines, shown as below). In the following example, the spot.x channel will return α degree for spot A, β degree for spot B and γ degree for spot C, so the input results should be $[\alpha, \beta, \gamma]$, approximate [45, 130, -70]

The spot.x test spots in horizontal space whereas the **spot.y** channel test spots in vertical space



spot.x channel example

The **spot.d** channel will return distances between the agent and the spot in 3d space, like the following picture, the spot.d channel will return the input results [d1, d2, d3] from current agent.

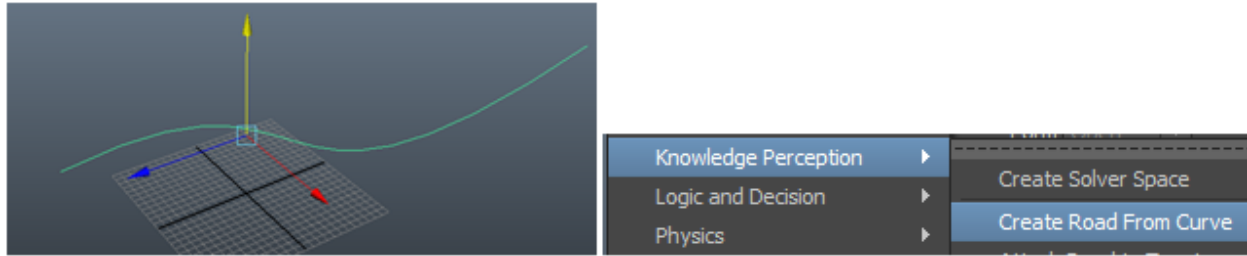


spot.d channel example

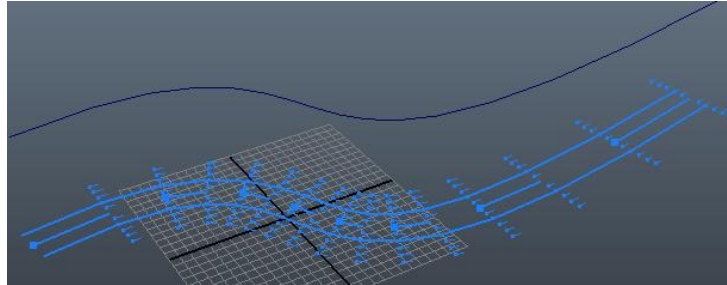
Road

Road is a main perception content of Miarmy. It's actually a point in 3d space. With spot channels, agents can test distances and directions with the spots and interactive with them, such like follow a spot, or trigger some behaviors near a spot.

For creating road, just select any curve, and click:



Creating road from curve

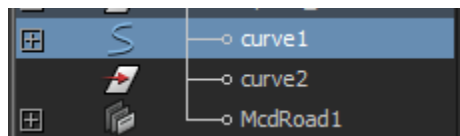


Road under the hood

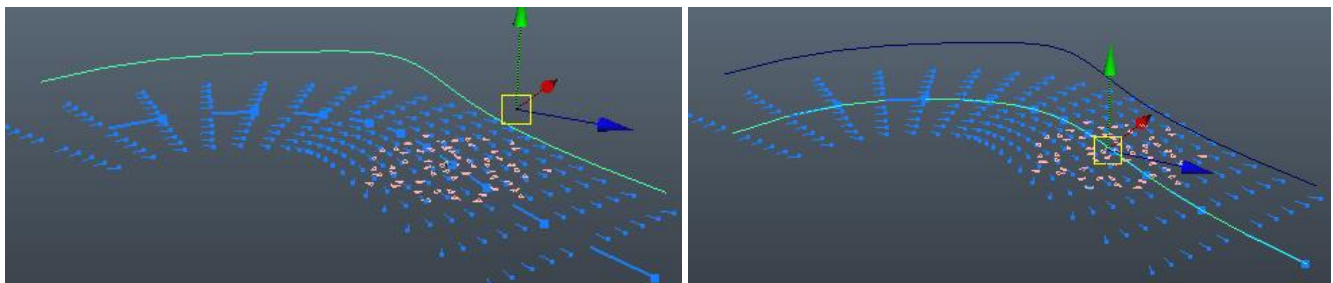
Assets

Once we create a road, there are 3 kinds of asset been created, they need working together for displaying the road and interacting with agents. Please manage them well and don't delete any one of it, or your road would not working properly.

1. Road control curve
2. Road mapping curve
3. Road node



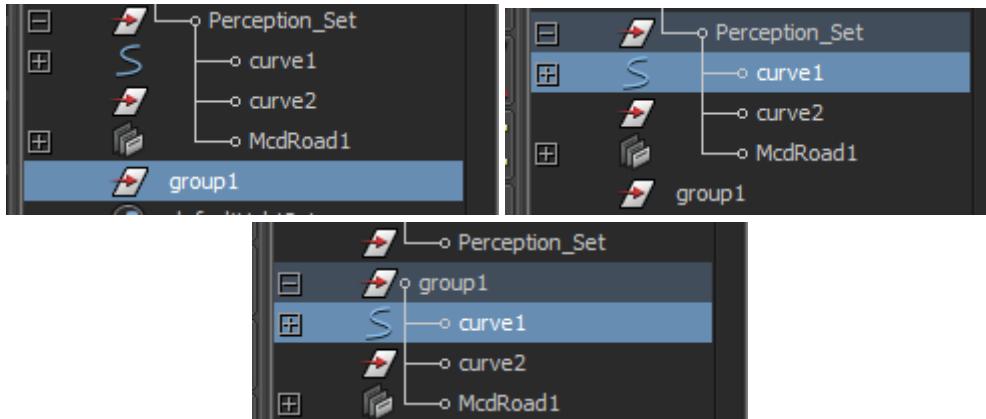
Take a look at the following pictures, in the left picture, the selected curve on top of road is the “control curve” and the blue-arrows-formed road on the ground is the “road node”. In the right picture, the selected curve on the ground is the “mapping curve” which is a mapped curve from control curve. It always been set as **Intermediate Object** (know more about Intermediate Object, please Google “Maya intermediate object”).



Control curve and mapping curve, the blue flow arrows are road node

Management by Parenting

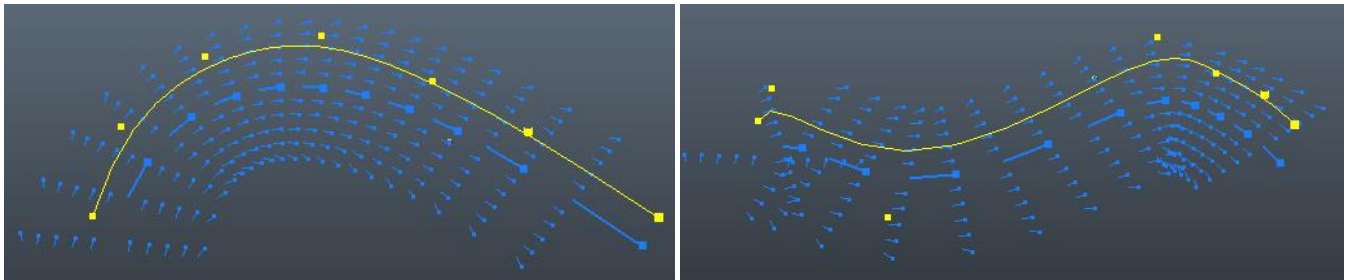
For parenting road to others node ,you just need parent the control curve to other node. For example, you need parent your road from “Perception_Set” to “group1”, you just need select curve1 and parent curve1 to the group1, then the rest suff will be automatically moved to group1.



Parenting “road” process from Perception_Set to group1, only operate on curve1 (the control curve)

Road Visualization by Control Curve Points

Arrange well the points on control curve can lead better visualization of the curve, just like the pictures below.

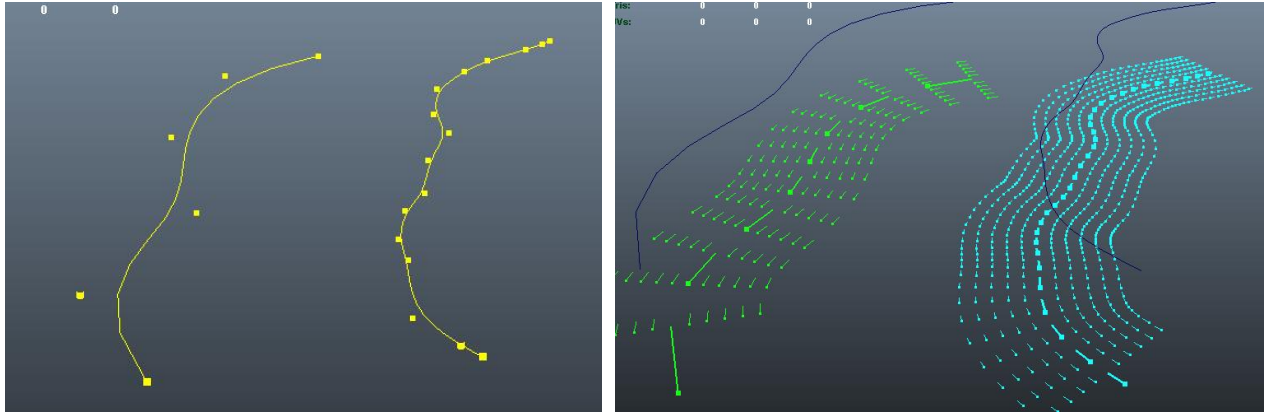


(left:) A well arranged control curve (right:) a non-well arranged control curve

But please don’t add or delete points on control curve, because in fact the control curve is controlling the “mapping curve” all the time. The road will not working if you add/delete the topology of the control curve.

Road Precision by Control Curve Points

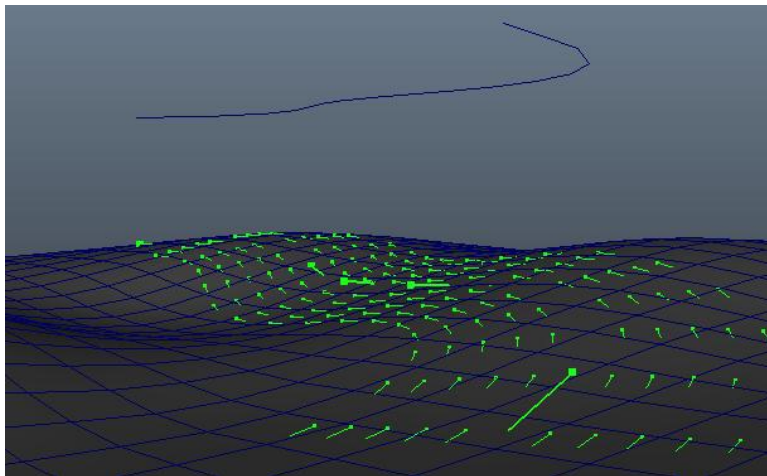
The precision of road is determined by the complexity of control curve (before creating)



Usually, a more precise road only can lead better visual result, but the calculation is slower, we not recommend you do that.

Attach Road to Terrain

Road can be attached to terrain. Select the control curve and terrain mesh, and click Miarmy > Knowledge Perception > Attach Road to Terrain, your road will be attached to terrain.



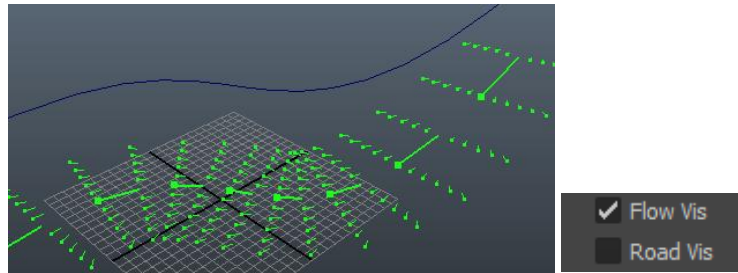
Road has been attached to terrain

Road Mode

Road can be 2 modes: flow mode and road mode.

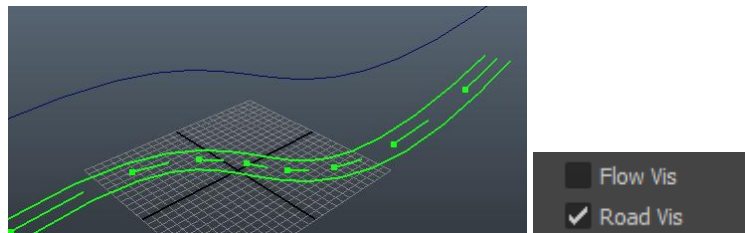
- Using flow mode, you can roughly constrain a group of agents follow a flow and make them convergence or separate by flow edge direction.
- Using road mode, you can precisely constrain you agents in road even a portion of road.

Flow mode:



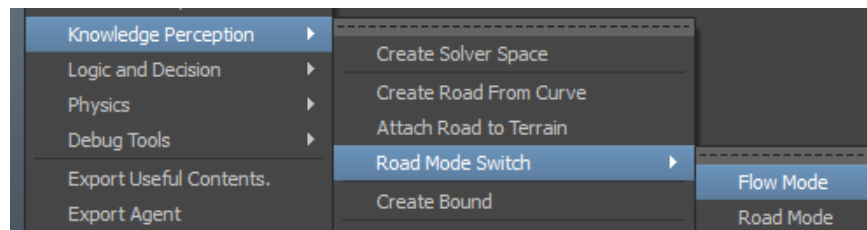
Flow Mode: turn on flow and turn off road

Road mode:



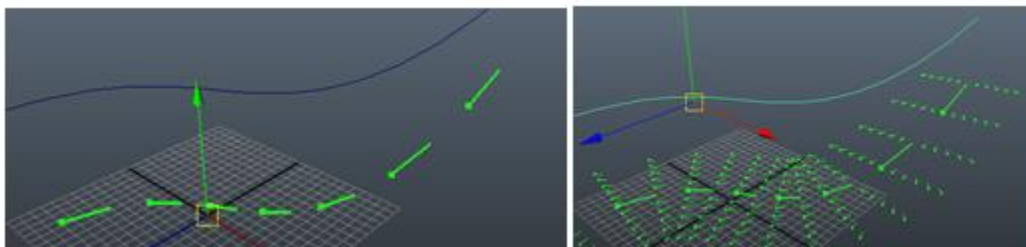
Road Mode: Turn on road and turn off flow

Usually, you can switch mode by menu item:



Switch road mode in menu item

Note: There would be an update problem here. Sometimes, after switching mode, the road would disappear, like this (picture below, left). You can simply move the control curve little bit, it would be ok. (Right picture)



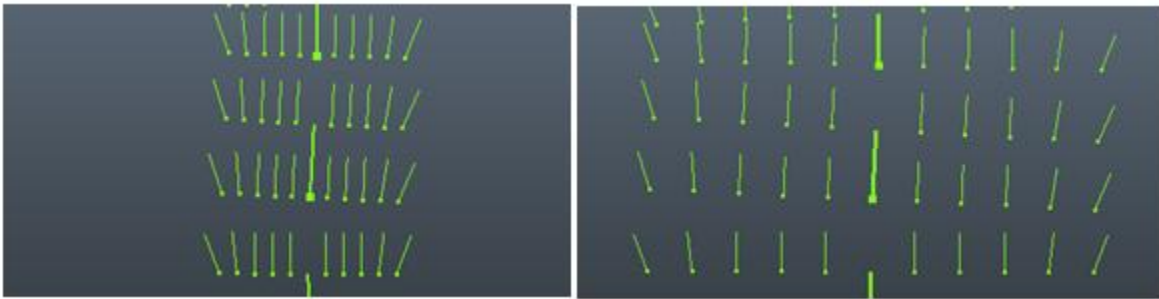
Flow Mode Attributes

Please notice, flow mode is not support indexing techniques,

Flow Width	697.3
Flow Orient	35
Flow Edge	0.3
Flow Len	80.8

Flow mode attributes

Flow Width Attribute:



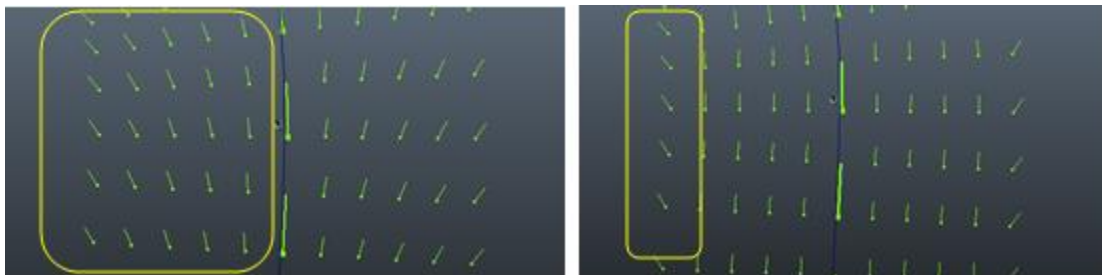
The narrow and wide flows, controlled by “flowWidth” attribute

Flow Orient Attribute:



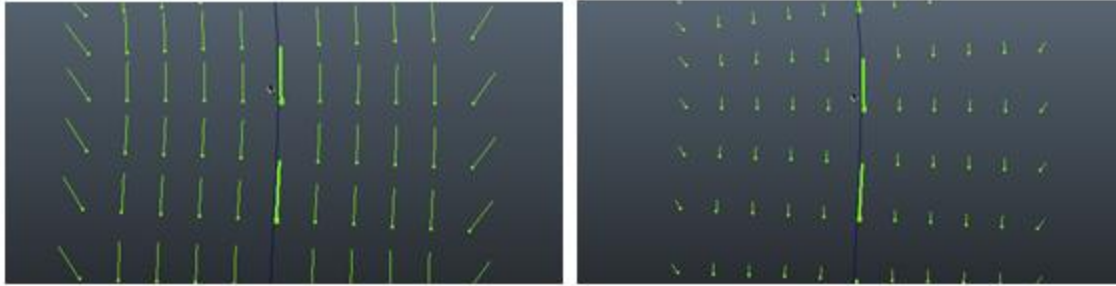
The outward and inward flows, controlled by “flowOrient” attribute

Flow Edge Attribute:



(left:) 1.0 (100%) edge ratio (right:) 0.3 (30%) edge ratio

Flow Len Attribute:

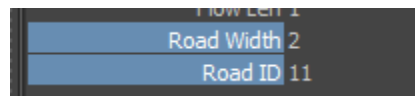


The long and short arrows' length of the flows, controlled by "flowLen" attribute

When the agent is in flow, it can feel it automatically.

Road Mode Attributes

Road mode can use indexing techniques.



Road mode attributes

Road Width Attribute:

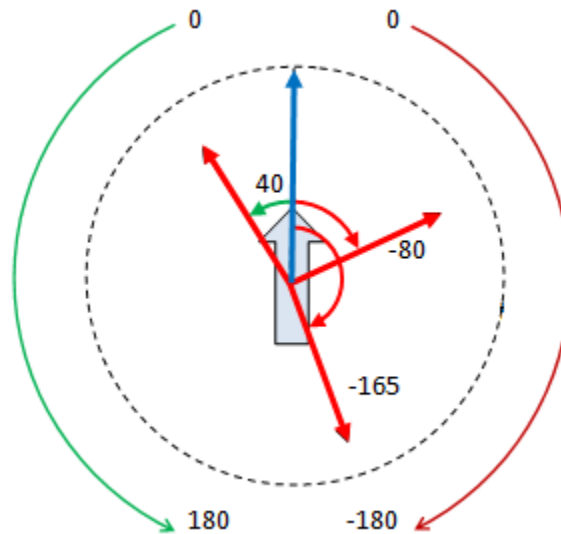


Narrow or wide roads, controlled by "roadWidth" attribute

Road for perception

Road Orientation

No matter road or flow mode of the road, the relationship of road orientation and agent orientation is the same, just as the picture shown below. If the road direction pointing to the agents left, the return value should be positive, if the road direction pointing to the right of agent, the return value should be the negative.



The relationship of road orientation and agent orientation (both road and flow mode, the same)

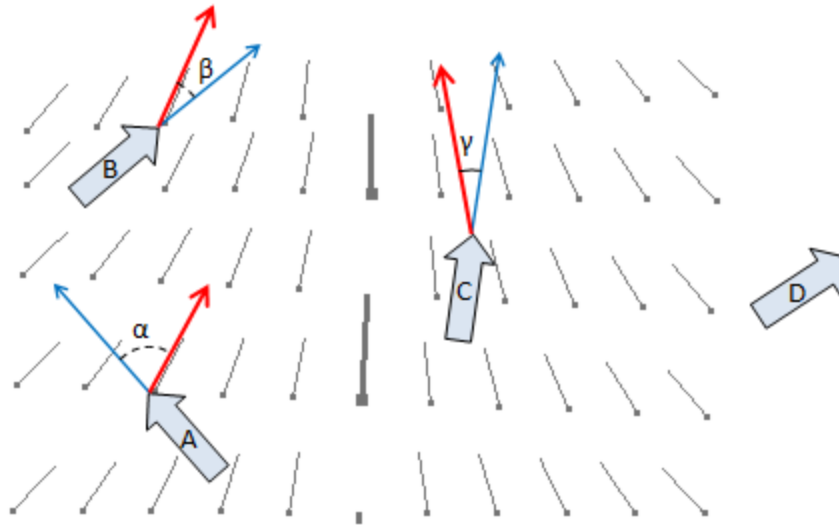
Road Channels

Different mode should use different channel, or they will not work.

- If you are using flow mode, there is only one channel called `road.flow` available.
- If you are using road mode, you can use `road.x` and `road.ox` channels.

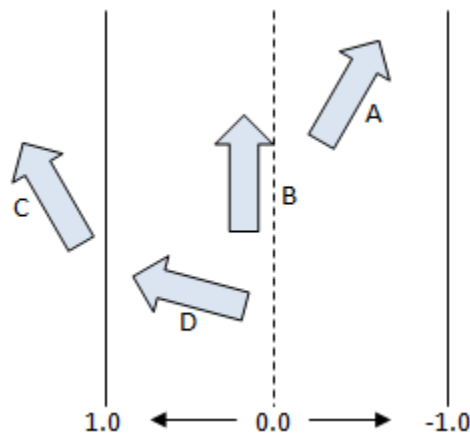
Firstly, if the agent isn't on the flow, the channel will return blank. For the agents on the road, the channel will return the angle between the agent Z-axis and the flow direction.

Take a look as the following picture, the red arrows are the flow direction, and the blue arrow are the agent Z-axis. For agent D, the channel `road.flow` will return blank, for agent A, B and C, the channel will return $\alpha = -70$, $\beta = 30$ and $\gamma = 25$.



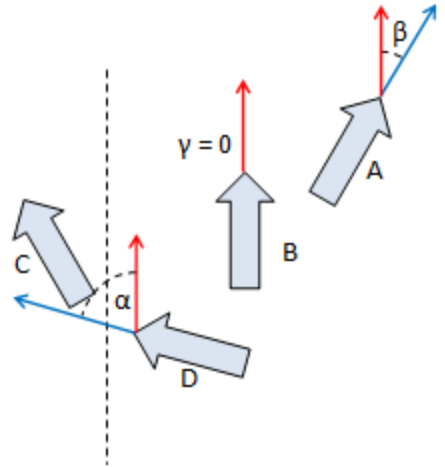
road.flow channel example, red: flow direction, blue: agent z axis

The `road.x` channel will return the position of agent on the road, the middle of the road is 0, and the left and right side of the road are 1.0 and -1.0. In the following example, the agent C will return blank because it's outside of the road, the agent A, B and C will return -0.4, 0.2 and 0.5 respectively.



road.x channel example

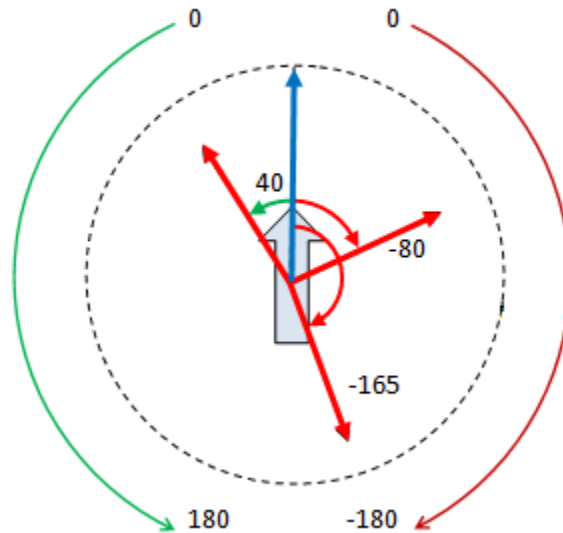
The `road.ox` channel will return the angle between the agent Z-axis and the road direction, the red arrow is the road direction and the blue arrows are the agent Z-axis. For the agent C, the channel will return blank, and for the agent A, B and D, it will return $\beta = 35$, $\gamma = 0$ and $\alpha = -70$ respectively.



road.ox channel example, similar with the road.flow

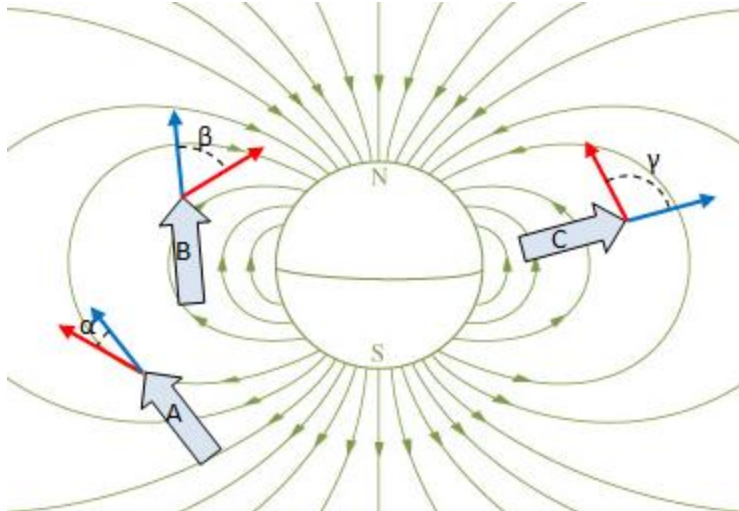
Maya Field

Agents can feel Maya field directly without any additional setup.



Red arrows are the directions of field and the blue arrow is the Z-axis of agent

Similar the road.ox, the **field.ox** channel will return the angle between agent Z-axis and the field direction of agent position in horizontal space. In the following example, the channel will return result $\alpha = 25$, $\beta = -65$ and $\gamma = -110$ respectively, for agent A, B and C.



Arbitrary Maya field and field.ox channel example

The field.ox is dealing with horizontal space whereas the [field.oy](#) is dealing with the vertical space

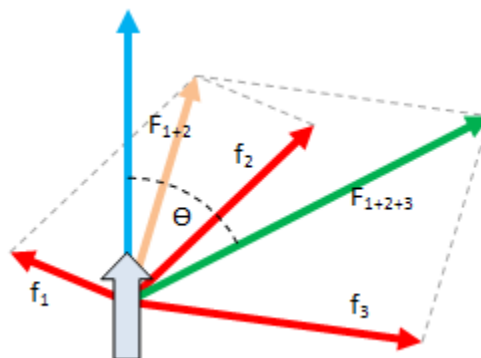
The [field.a](#) channel can return the Maya field amplitude from the agent position.

Composition of Forces

The force field related channels, like the **Maya field**, **fluid**, and the **wind**, are not support indexing techniques. But they support composition of forces.

Imagine that there are several fields in scene. At this time, channels like the field.ox will **NOT** return an array contain each result from each field. Instead, it will return only one result based on the composition of force field.

Like the following example, the 3 red vectors are the initial forces from 3 fields. The orange vector is the composition force of f_1 and f_2 , and the Green vector is the final composition force vector from f_1 , f_2 and f_3 . Channel field.ox will return angle between the blue vector (Z-axis of agent) and the green vector (Force Composition), which is angle Θ .



Composition of forces $f_1, f_2, f_3 \Rightarrow F_{1+2+3}$

Maya Fluid

The fluid channels are totally the same as the of Maya field channels, because in fact, fluid is somewhat like the field, we can read field information from fluid, like the direction and the amplitude.

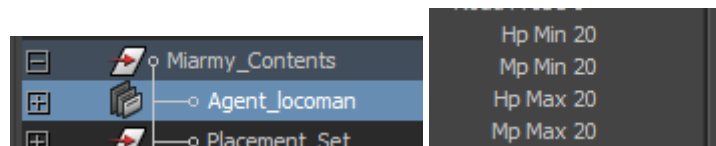
So the fluid channels including, please refer the field perception contents.

- **fluid.ox**
- **fluid.oy**
- **fluid.a**

HP & MP

HP (hit point) and MP (mana point) are 2 custom variables for arbitrary usage, usually be used to simulate the energy of agent. Each one of agent has some initial HP and MP value after first creating. This value is store in memory and only can be modified by the related channels.

You can set the initial HP and MP in McdAgentGroup node and randomize them when initializing.



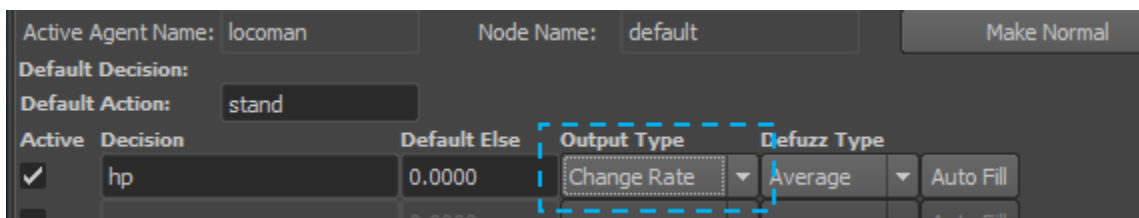
The McdAgentGroup node and the attributes on it

The **hp** channel for **input**, it will return the current value of HP inside the agent memory.

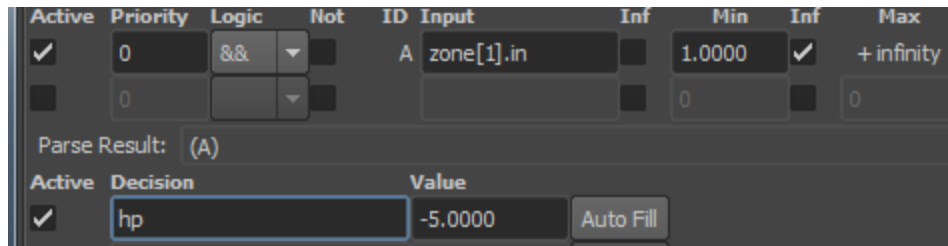
The **hp** and **hp.set** channel for **output**, it will set the value of HP to the specific value.

In most of the cases, we are not use using hp channel like that. Usually, we firstly setup the hp channel in default decision node, make the output type “change rate”.

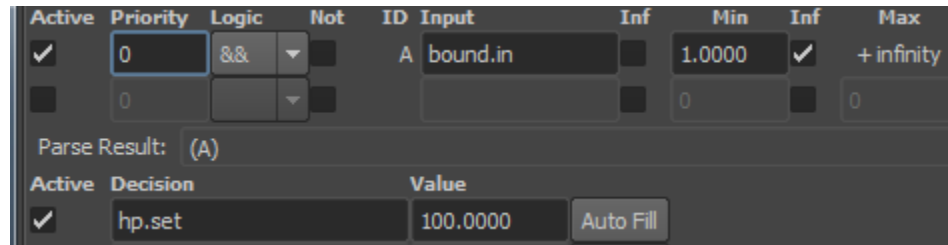
At this time, the **hp** for **output** channel will not set HP value, instead of that, it will change the HP value per second. For example, when the agents are walking on the radiation zone (zone[1].in channel greater than 1), the HP will drop some points(5 points) per-second. And if you want to reset the HP value in agent memory, you need use **hp.set** channel because at this time the hp channel is in “change rate” mode, you cannot reset hp value directly by hp channel.



Setup output type to “change rate”



Agent hp will drop at speed of 5points/second if it in the zone which index is 1



Reset agent hp to 100 if the agent in bound

The MP and the mp channels are totally the same as HP and hp channels

Scene Info

The channel **frame** will return the current frame number.

The channel **goFrame** will return the frame number since beginning of simulation start.

Randomize

The noise channels will return a random number from 0 to 1 based on different conditions. You can easily combine this value with fuzzy logic to make your output randomize.

Channels

The channel **noise.id** will return a random number based on agent ID. Each agent has a unique ID in scene once it was placed out.

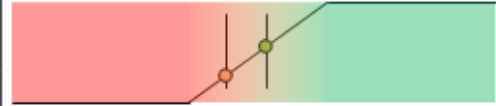
The channel **noise.time** will return a random number based on current frame number.

The value range of the returned random number is from 0 to 1, a float point number.

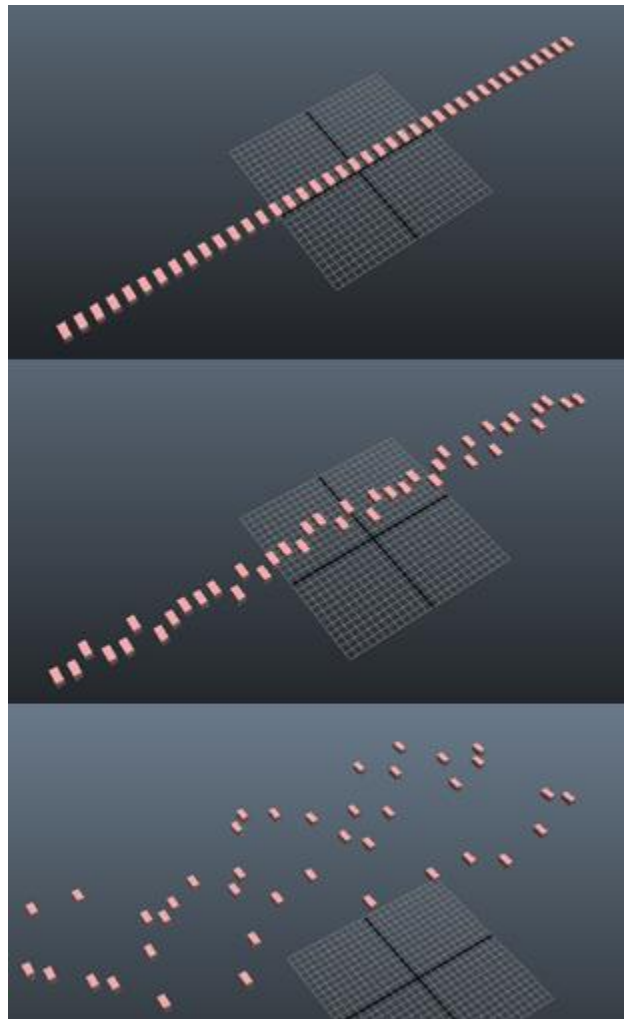
Usage Example

Randomize speed

Input	Inf	Min	Inf	Max	FuzzyIn	FuzzyOut
noise.id	<input type="checkbox"/>	0.5000	<input checked="" type="checkbox"/>	+infinity	0.5000	
	<input type="checkbox"/>	0	<input type="checkbox"/>	0	0	0



The bigger activated, the faster the speed, random by agent id



Random speed

Randomize actions



Random action

Simulation Space

World Transformation for physical simulation

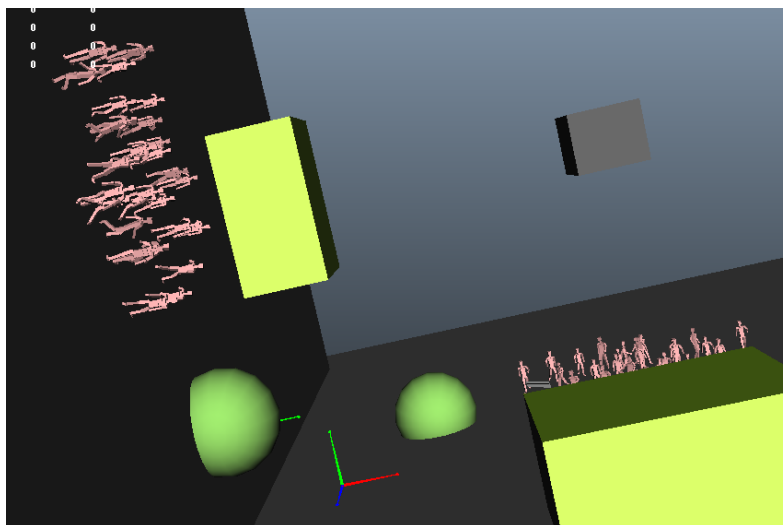
If our agent enable dynamics, no matter where the agents and how the structure like, the agent will enable ragdoll in world transformation space. Including the kinematic primitives, terrains, all of them will join the world space calculation.

Local Transformation for non-physical simulation

Except physical simulation, in most case and more interesting, we are dealing with the agent which driven by logic and decisions.

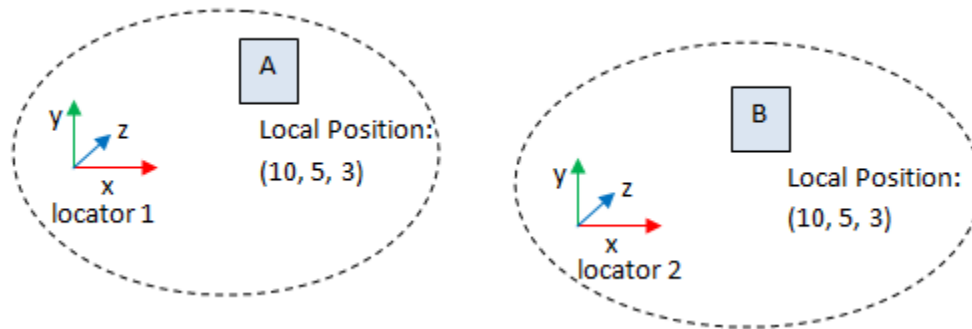
As our design, all of the Miarmy agents (not enable dynamics) and perception contents are calculated in local transformation. Our engine will fetch the data such as the rotate of agent, the position of spot, all from their local transformation.

So you can easily parent you agents to different transform nodes such as locator nodes make engine calculate them separately. With this feature, you can easily simulate different groups of agent on different moving objects like some battleships.



Simulation in different transformation space

But only using the simple transform node is not enough at all, because the agents in a transform node can feel the agents in others transform node. This may cause the calculation error. Take a look at the following example, we were going to make them moving in different space and not affect each other. However, their local positions are the same. When the engine calculating, the agent A can feel agent B and try to avoid it, also the agent B will avoid agent A.



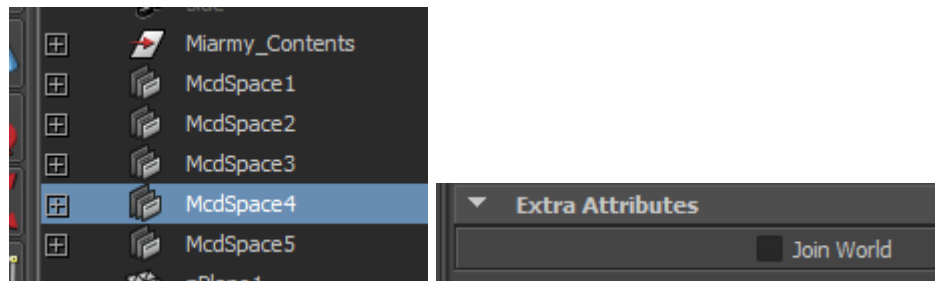
Same local positions in different transform nodes

For entirely avoiding agents in one space feel the agents in another space, you can group them to **Solver Space Node**.

Solver Space Node

Technically, the solver space node is the same as locator (without shape node), also a kind of transform node. But our engine can recognize it and calculate the agents and contents inside of it independently.

For creating solver space node, you just need simply click Miarmy > Knowledge Perceptions > Create Solver Space.



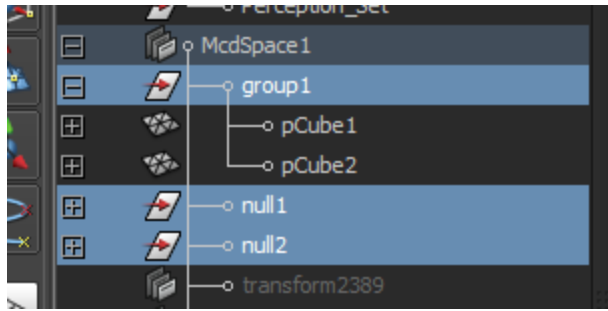
(Left:) the solver space node, (right:) solver space attribute flag

Once you putting the agents and perception contents inside of solver space node, they will work independently in local transform space. The agents in a solver space node will not feel the agents or perception contents in the others solver space nodes.

If you want to parent you agents to a solver space, please parent the place node to the solver space or using inverse place for recording parent's name.

The solver space hierarchy

The contents in solver space can be parented to any other nodes inside of this solver space node, just make sure the root of them is this solver space. Shown as the following picture, McdSpace1 is a solver space node, you can put different perception contents in different groups in McdSpace1, some spots in “null1”, and some roads in “null2”. Our engine can recursively find out the solver space node.



Arbitrary hierarchy under Solver Space node

Join World Attribute

There is a special “joinWorld” attribute on each of solver space.

Once you enable this attribute, the engine will take the world transformations of agents in this solver space node instead of normal local transformation. And the agents in this solver space can feel others agents and perception contents in the others solver space nodes which “joinWorld” attribute been also enabled. Calculate just like they are in world space, but they are actually also the child of each solver space node.

Output Behavior Channels

Transform Speed

- **tx:** move some units per second in the x-axis of the agent object
- **ty:** move some units per second in the y-axis of the agent object
- **tz:** move some units per second in the z-axis of the agent object
- **rx:** rotate some degrees per second by the x-axis of the agent object
- **ry:** rotate some degrees per second by the y-axis of the agent object
- **rz:** rotate some degrees per second by the z-axis of the agent object

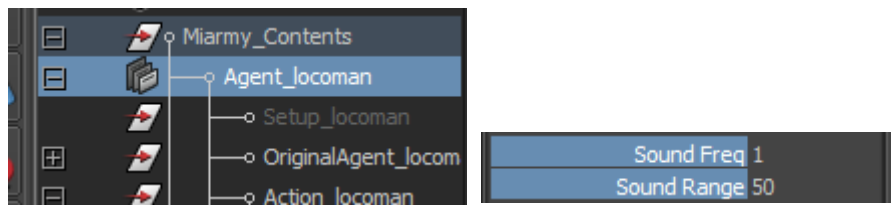
Bone Offset

- **<boneName>:tx** add offset units in translate X of the bone whose name is boneName
- **<boneName>:ty** add offset units in translate Y of the bone whose name is boneName
- **<boneName>:tz** add offset units in translate Z of the bone whose name is boneName
- **<boneName>:rx** add offset degrees in rotate X of the bone whose name is boneName
- **<boneName>:ry** add offset degrees in rotate Y of the bone whose name is boneName
- **<boneName>:rz** add offset degrees in rotate Z of the bone whose name is boneName

Sound

The sound has two channels can be output, they are **sound.f** and **sound.a**.

By default the frequency and range of sound are the values from McdAgentGroup node. However, you can use the sound output channels to modify them.



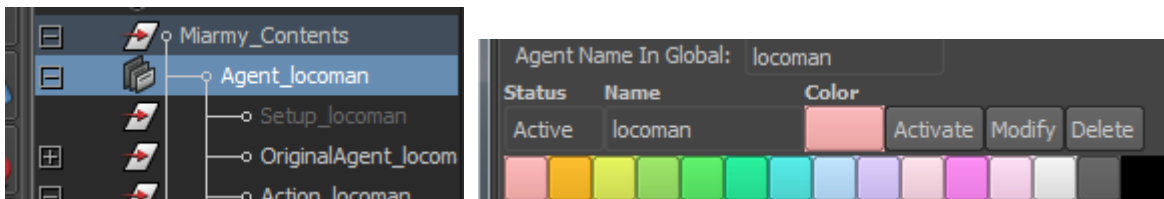
(Left :) the McdAgentGroup node; (right :) The default sound Frequency and range on node

Note: if the sound range is 0, means the sound range will take the value calculated from bounding box, see **Part 4 Agent Infrastructure – Original Agent**

The channel sound.f in output channel can modify the sound frequency of the agent, whereas the channel sound.a in output channel can modify the sound range.

Vision

There is no vision channel for output directly, but there is a **color** channel can modify the result of vision received.



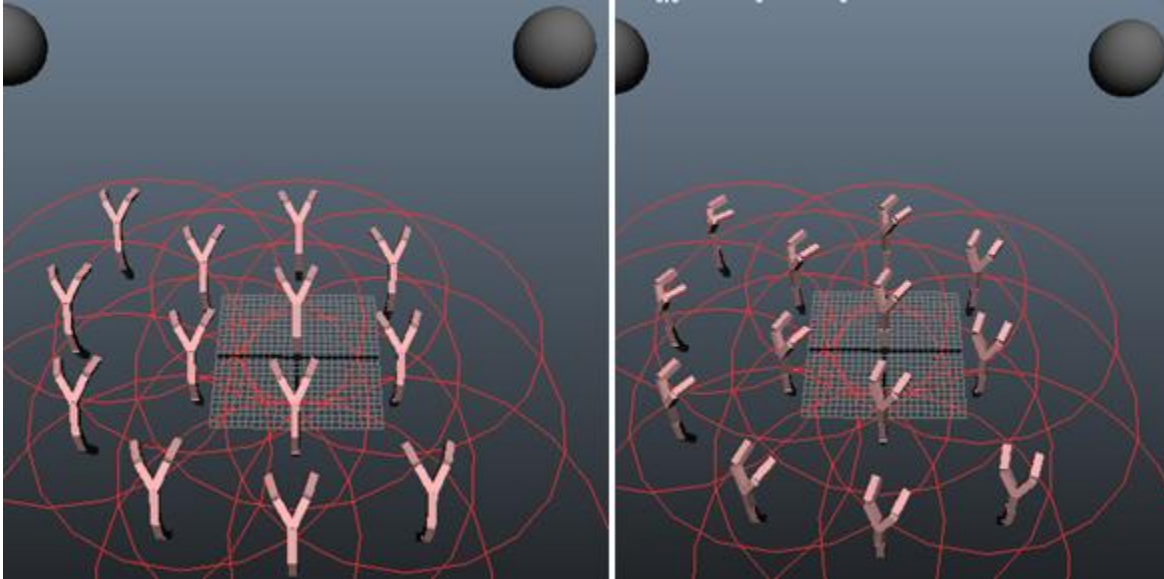
(Left :) the McdAgentGroup node; (right :) setup the color for this type of agent

The channel color in output channel can modify the color of agent, and change the vision.h results.

If the color result is **-1** in output, the agent cannot be seen by any other agents.

Aim Constrain

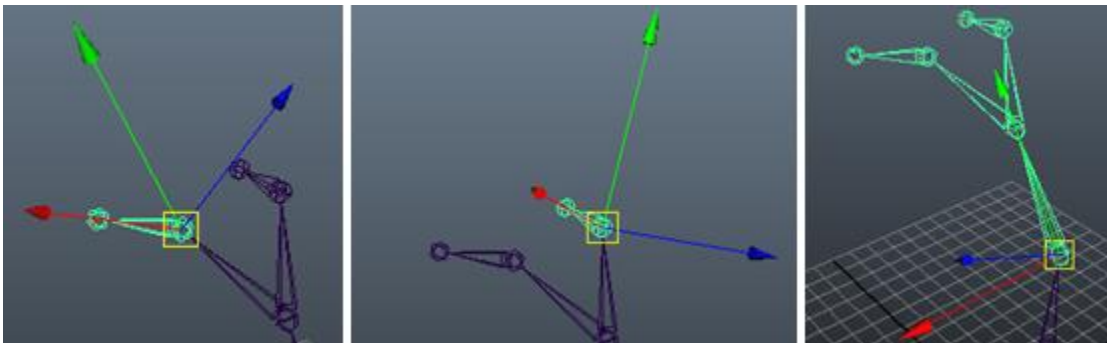
We provide a turn-key aiming feature for the bones of agent, which can drive your agent bone aim to target. There are some subtle features of it.



Aim channels

Preparation for Bone Axis

Before generating the original agent, we need setup our rig. If we want to one of the bone aim to target, we need firstly make sure the aiming axis and up vector. In the following pictures show, before generating original agent, please make sure the X (or Z) axis is the aim axis and the Y axis is the up vector. (Note: the other orient scheme is not support for current version of Miarmy)



The X (or Z) should be aim axis and the Y should be up vector

Aim Channel Features

Aim Channels

The channel conventions of aiming:

<Bone Name>:aim<axis><space>-><Target object>

<Bone Name>:aim<axis><space>(<percentage>)-><Target object>

- The <Bone Name> should be the segment bone name from agent memory. You can check the real name in agent memory by Miarmy > Agent Viewer
- The <axis> should be the axis of bone which you want to it aim to target
- The up vector should be always Y axis (for this current version of plugin)
- The <Target Object> name should be the unique name in Maya Scene. The dag nodes with the same name in different hierarchy many cause problems.

For example:

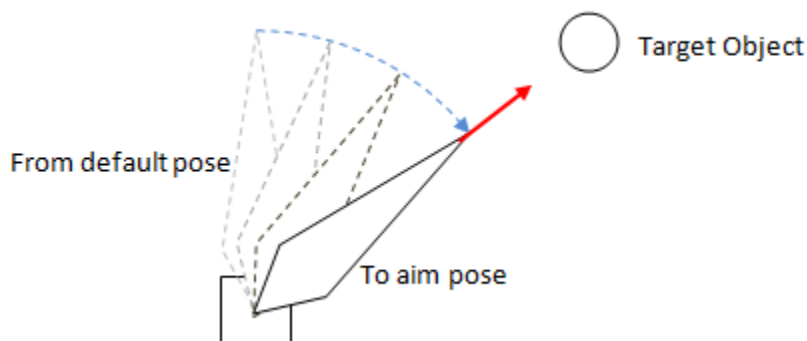
- **eyeL:aimX3d->pSphere1**
- **head:aimZ2d->pSphere1**

Once the channel is activated, the bone which you specified will turn and aim to the target with a rotation speed gradually. This speed is determined by channel result which range is 0 to 1.

Note: the aim turning process is interpolated by Quaternion, so there is no Gimbal Lock.

Gradual Aim

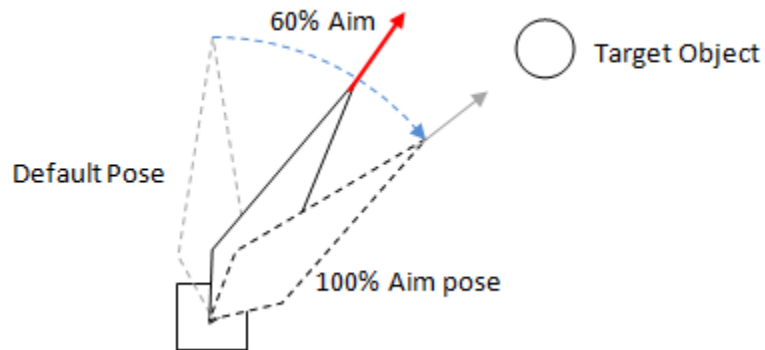
Once the bone wants to aim to target, it will aim to target gradually rather than pointing to target directly. Like the following picture, from default pose, the bone point to the target object gradually, and the speed is depending on the output value. 0 make the bone still, 1 make the bone instant point, the value between 0 and 1 (such as 0.4) make the bone point to target 40% each frame.



Gradually aim to target from default to aim pose

Percentage Aim

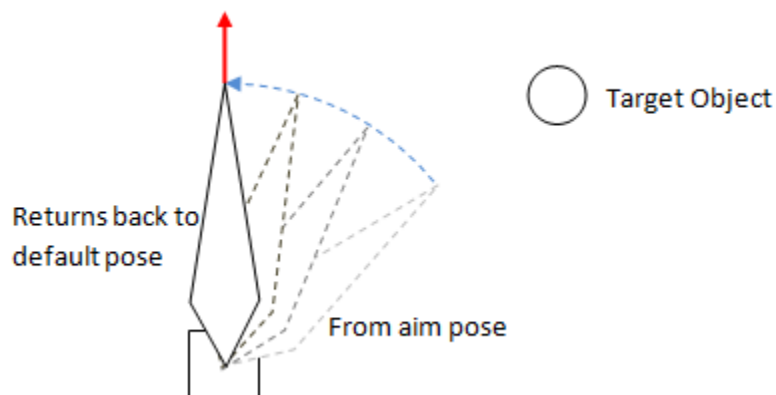
We can specify percentage in aim channel to make the bone aim to right direction but not fully aim to target. For example, we can specify a 60% aim channel like this, `eyeL:aimX3d(60)->pSphere1`, you may notice the bone will approach to the 100% aim pose but not fully aim to target. Finally, it will stand the pose which performs 60% aim direction between the default pose and 100% aim pose.



60% aiming make non-full aim result

Self-restitution

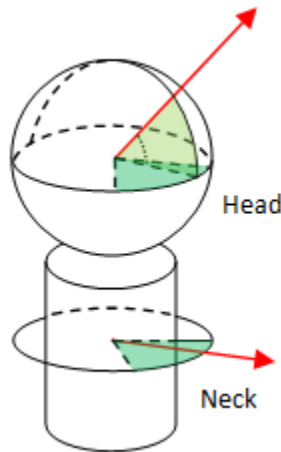
Once the sentence is not true (the default output value of each aim channel is -0.2) or the channel result is negative, the bone will restitute to the default pose gradually and automatically.



Returns back to default pose automatically

Aim Space

Aim space is simple. In some case, we just want to the bone aim to target in horizontal space, such as our “neck”. Our head can rotate to any directions but our neck can only rotate in some degree in Y axis:



Use 3d for the head whereas 2d for the neck

Hierarchical Aim Mechanism

This process is entire automatically, and here we just explained what is happen under the hood.

When you have multiple bones in the same hierarchy want to aim to the same (or different) target, the order of aim bone may cause problem. Imagine that your hand and arm want to point to the some targets. If your hand point to the target firstly, and then you rotate your arm secondly, after your arm aligning to its target, maybe the previously done hand will not point to right direction.

With Miarmy hierarchical aim mechanism, we firstly collect all aim tasks and then execute them hierarchically. For example in the arm tree, it's definitely arm will aim to target before the hand performing aim.

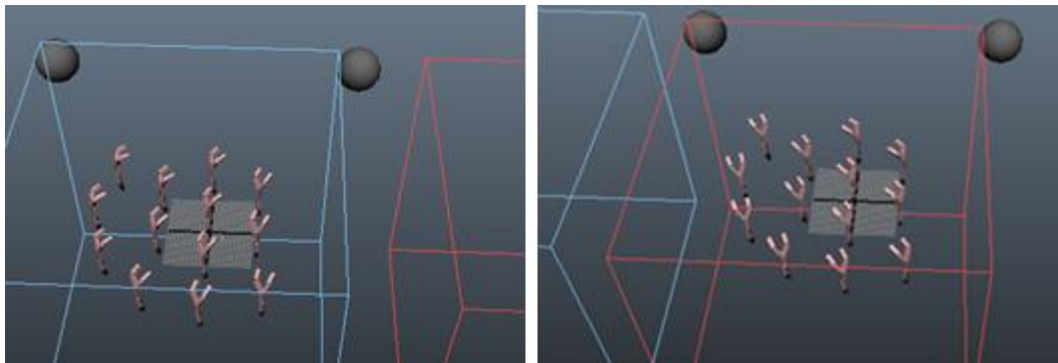
Multi Targets Choose

Rather than Maya traditional aim constrain, we can make the agent bone to aim arbitrary number of targets, just using simple sentence active:

Let's take a look at the following example directly.

We want to our agents aim to right sphere when they are in blue bound whereas aim to left sphere when they are in red bound. Just using the bound channels and the indexing technique, we can easily achieve that.

- head:aimZ2d->pSphere2 (active in red bounding box)
- head:aimZ2d->pSphere1 (active in blue bounding box)



Active	Priority	Logic	Not	ID	Input
<input checked="" type="checkbox"/>	0	&&	<input type="checkbox"/>	A	bound[0].in
<input type="checkbox"/>	0		<input type="checkbox"/>		
Parse Result: (A)					
Active	Decision	Value			
<input checked="" type="checkbox"/>	eyeL:aimX3d(50)->pSphere2	0.8000	Auto		
<input checked="" type="checkbox"/>	eyeR:aimX3d(90)->pSphere2	0.8000	Auto		
<input checked="" type="checkbox"/>	head:aimZ2d->pSphere2	0.2000	Auto		

Active	Priority	Logic	Not	ID	Input
<input checked="" type="checkbox"/>	0	&&	<input type="checkbox"/>	A	bound[1].in
<input type="checkbox"/>	0		<input type="checkbox"/>		
Parse Result: (A)					
Active	Decision	Value			
<input checked="" type="checkbox"/>	eyeL:aimX3d(50)->pSphere1	0.7000	Auto		
<input checked="" type="checkbox"/>	eyeR:aimX3d(90)->pSphere1	0.7000	Auto		
<input checked="" type="checkbox"/>	head:aimZ2d->pSphere1	0.1000	Auto		

2 different targets can be triggered aim by different conditions

The bone will always aim to the target which makes the output the most activated.

Action

We will talk action related channels in details in **Part 6 Action**

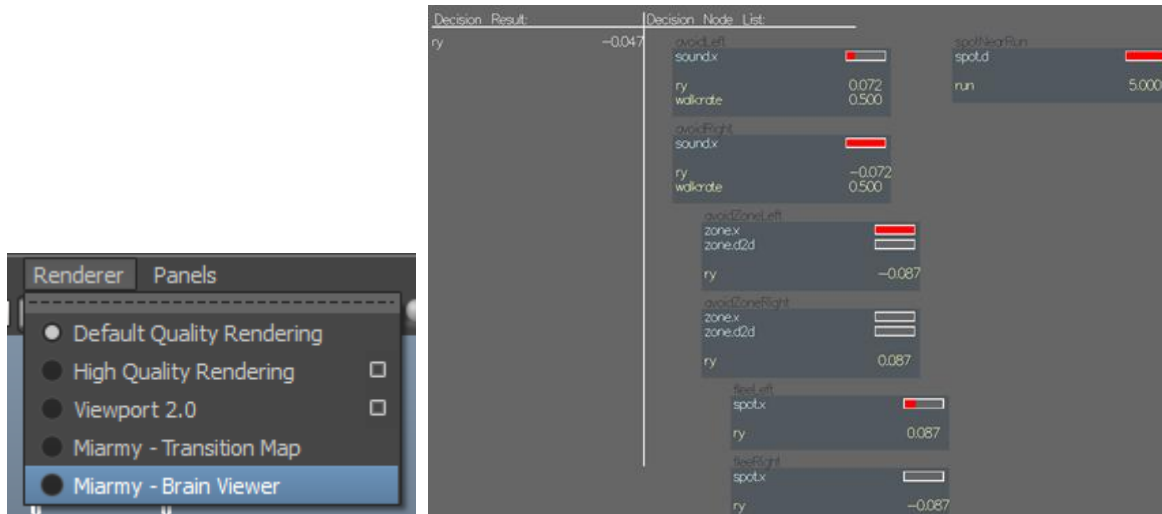
Ragdoll

We will talk this in details later, see **Part 7 Physical Simulation**

Watch Logic Results

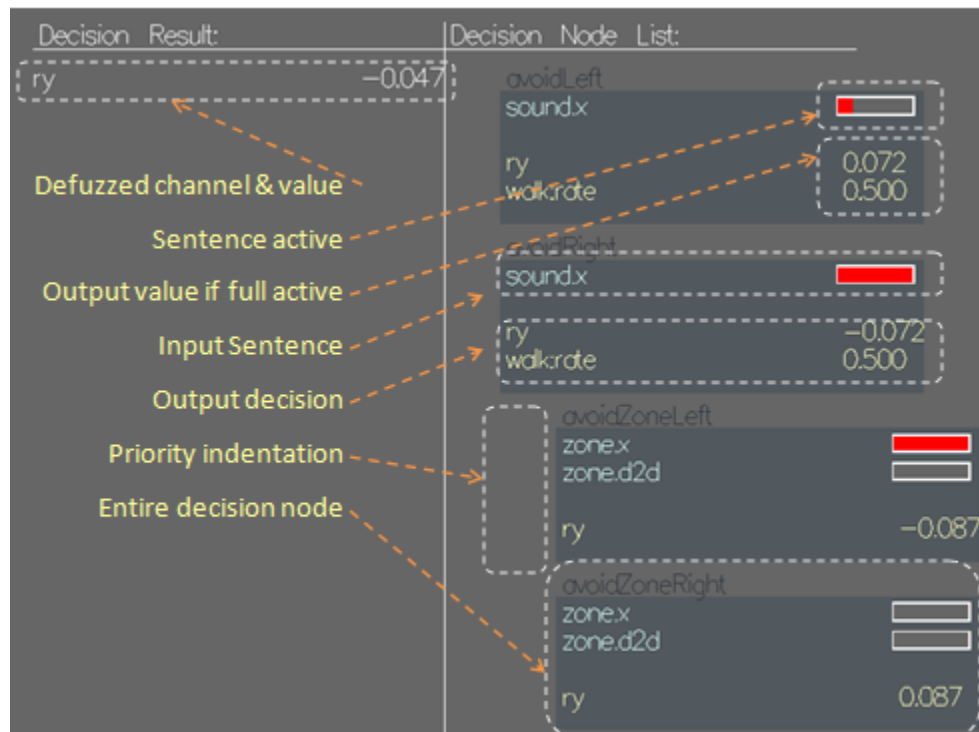
Brain Viewer

Using brain viewer, you can check the agent logic status and easy for debugging your agent brain.



Brain viewer GUI

Please check out the features demonstration picture below, from brain viewer, you can easily watch the brain logic structure and sentence active status, and their priorities and defuzz behavior results.



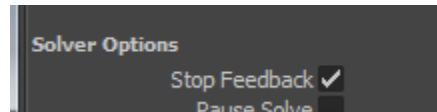
Features details in brain viewer

The brain viewer should be watched in Top View. If the display not correct, please crank the near/far clip plane for trying to correcting it.

Feedback in Essential

Feedback mechanism is an important feature of Miarmy Engine, when simulation, the engine can always feedback the logic and action status from “marked” agent. For marking the agent, you need just simply select any one of agents. The latest selected agent will be marked automatically. Our engine will do an extra time of calculation for this agent and feedback the result to Brain Viewer and Transition Map.

For blocking the feedback, you need disable feedback feature of Engine in Miarmy >Miarmy Global. Once disable the feedback feature, the simulation will speed up about 20%. So we highly recommend you disable this after debugging agents and populating huge number of agents.



Block the feedback from agent

Part 6 Action

In this part, we will detailed explained how Miarmy Animation and Action system working and under the hood.

Action Infrastructure

Action Node Data

The action node contains

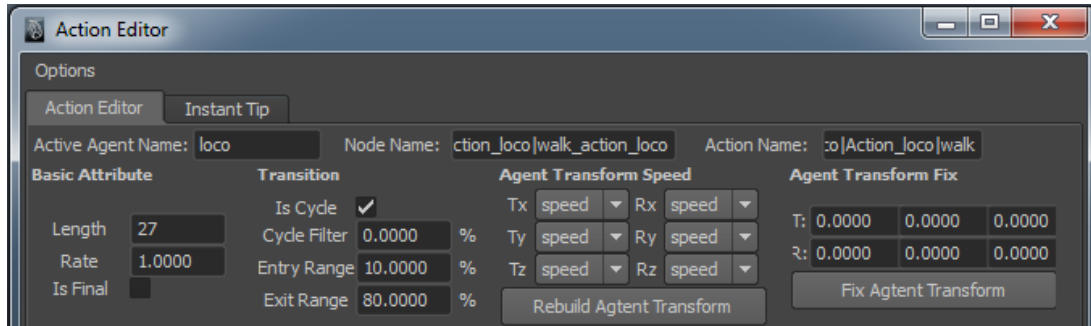
- **Action length (integer)**
The length of this action
- **Action Data (2D array)**
The pose data of each frame, using this data array and a frame number, we can set the agent pose. Note: the root data in this action data array would be modified after generating of agent transform data.
- **Original Root Bone Animation Data (array)**
Extract the root bone pose data of each frame from Action Data before generating agent transform data
- **Agent Transform Data (array)**
Calculated agent locomotion data for each frame from Original Root Bone Animation Data

In this version of Miarmy, the data of actions are stored according to the bone hierarchy. So, the actions drive agents by bone hierarchy. The bone structure of rig and original agent should be the same all the time for the same type of agent. The actions are generated from rig, and these actions can be shared to use among the different types of agents if the hierarchy of them are the same.

Note: in the next version of Miarmy, action drive agent support “hierarchy + bone name” combination method and coming in April. At that time, the action drive agent not only depending on the hierarchy, you can add some extra bones to the Original Agent and reuse the existed action node

Action Node Attributes for Playback and Transition

There are many attributes in each action we can setup them in action editor



Action Editor

Rate

The speed of playback for this action

Is Final

If enable, once agent transit to this action, it will never transit to others in any situation. Like the “dead” action.

Is Cycle

If enable, means we dealing with this action a cycle action can be transit to itself.

Cycle Filter

Only work with cyclic action, blending the end self-transition part to the head part of the action by some percent.



As picture above, after playing back green area, the action will perform self-cycle.

Entry Range

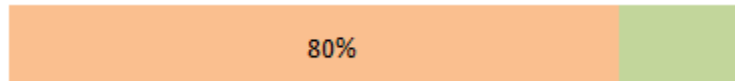
When previous action transit to current action, some percent should be smooth blended. This is the entry range of this current action. Like the picture below, when the previous action transiting to current action, 10% of action length should be blended from previous end to current start.



When transiting in, blend with previous action

Exit range

When current action should transit to next action, only when the current action playback after exit range percent, the current action can transit to next action, otherwise, maintains the self-playback



Only after 80% playing back, the current action can transit to other actions

Agent Transform Data

Agent Transform Data is generated from you “rig root”, and stored independently in action node. It's a kind of locomotion data can be applied to overall agent

As we all know, we create action from animation, it is a piece of motion in a range. For example a character is walking in scene and we build action data 24 frames.

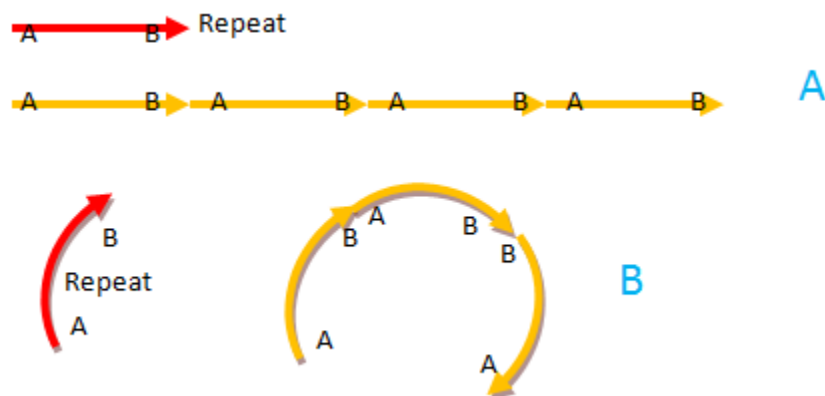
In the below picture, it's a walk cycle, no special:



A walk cycle from A to B, for example 24 frames

The normal animation will be from A to B and get back to A, once again, from A to B, repeatedly.

We want to apply it to the agents and make the agent move forward in any direction and not in one place. We want to make the agent travel in scene.



Transform “the walk cycle A to B” to “agent locomotion data”

The red line is repeatedly playback **animation**

The yellow line is the **action** with agent transform data. The movement of yellow paths shown is our purpose.

With agent transform data, we can apply the accumulate locomotion to the agent, and our agent can work independently in our scene

For this purpose, we need separated the animation in 2 parts:

- **Action data:** the animation pose data of each frame
- **Agent transform data:** the root node velocity data can be used to apply to single agent transform.

Usage:

1. **The action data part** is responsible to pose you agent **only**.
2. **The agent transform data part** is responsible to move your agent for each frame.

The agent transform data actually is an array of velocity value for each frame. So, no matter where the agent is and what direction the agent orient to, we just apply a velocity to it each frame. The result is, the agents can travel in scene anywhere respectively, and not repeat playback.

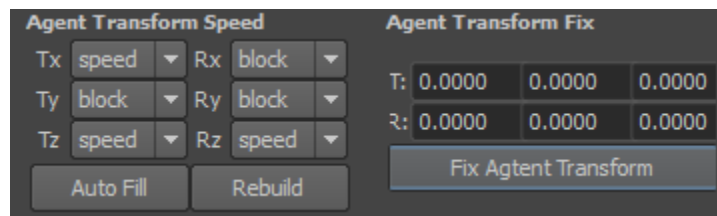
Agent transform data can be several types:

The “A” picture above shows the “locomotion” type, and “B” shows the “turning” type.

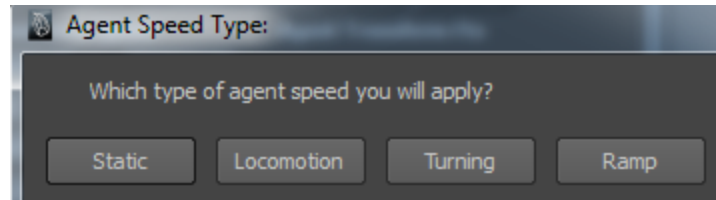
- **Static:** agent stands without transform change (e.g. stand, sit, cheer)
- **Locomotion:** agent walks in z direction straight (e.g. walk, stand to walk, jog, run)
- **Turning:** agent turning to left or right (turn left, turn right)
- **Ramp:** agent go up or go down (go upstairs, climbing)

Miarmy will automatically generate agent transform data for you when you create action.

Also we provide tools are able to re-build it or fix it



Click auto fill, you can build agent transform data as preset:

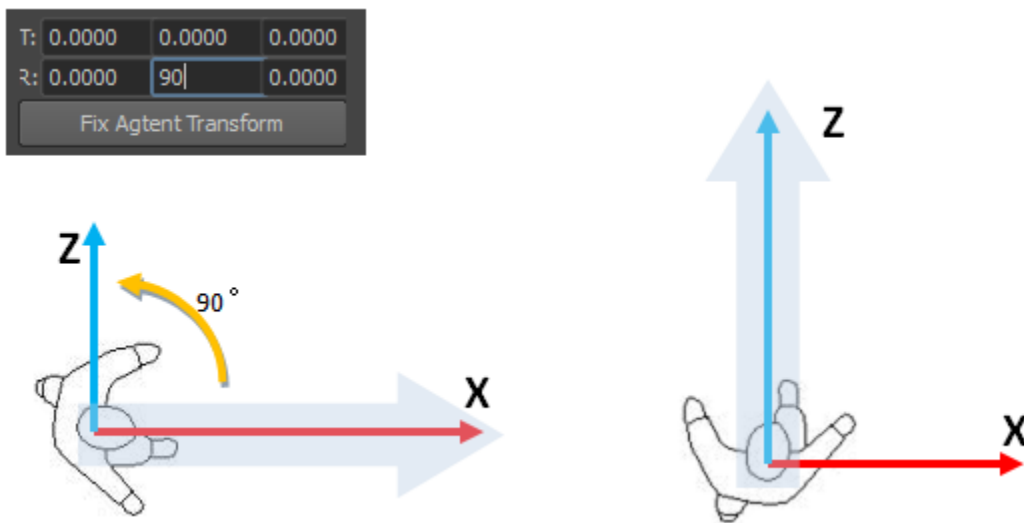


Fix Agent Transform is used for shifting or rotating your agent transform data:

For example1:

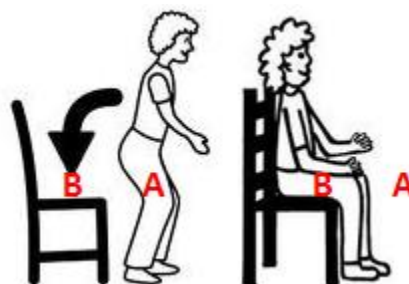
The standard agent orientation is "+Z", but if your animation rig is walking in x direction. After creating action, you can rotate it back to +Z.

Just fill 90 in action editor, and click "Fix Agent Transform"



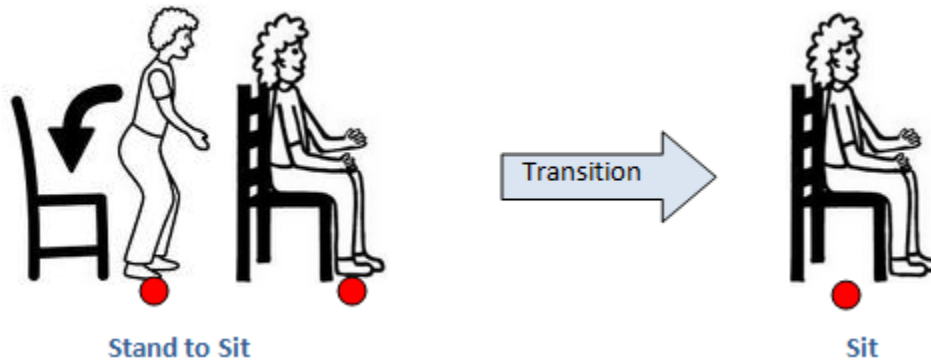
Fix the agent transform data

For example2 (sample 29 scene file)



The root of agent moved

Please notice the above animation, from stand to sit, character's root (hip) has been moved, from A to B. However, this action is a static action. When this action transition to sit action, there will be a weird behavior:

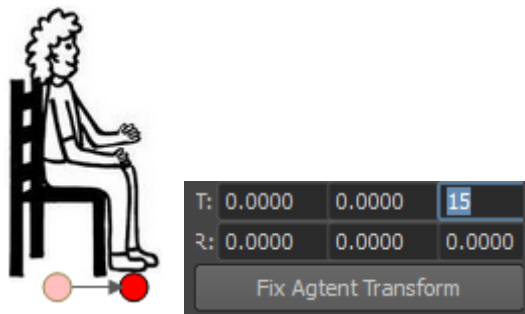


The red spot is the “origin”. The left action is “stand_to_sit” and right is “sit” action.

You may notice the

1. In “stand_to_sit” action: the Origin is in the character’s feet.
2. In “sit” action: the origin is in character’s root (hip)

If the agent transit from “stand_to_sit” to “sit”, the action will “shift” forward, because **the 2 action have different origin places**. At this time the only thing we need to is move the sit origin to his front, like the picture below:



Fix the agent transform data

The result will move the action entirely forward. The play back transition will now without “shift” and seamless.

Summary: Although the agent transform data is crucial, this data can be actually automatically generated. So, in most of time, you no need to care about that, just choose the right type. Everything will be fine.

Action Channels

Action Trigger

Trigger Action Name by Name

Logic: if the agent in the zone area of which the id is 0, perform walk

Active	Priority	Logic	Not	ID	Input	Inf	Min	Inf	Max
<input checked="" type="checkbox"/>	0	&&	<input type="checkbox"/>	A	zone[0].in	<input type="checkbox"/>	1.0000	<input checked="" type="checkbox"/>	+ infinity
<input type="checkbox"/>	0		<input type="checkbox"/>			<input type="checkbox"/>	0	<input type="checkbox"/>	0

Parse Result: (A)

Active	Decision	Value
<input checked="" type="checkbox"/>	walk	1.0000

Auto Fill

Trigger action directly by name of action

Trigger Action by Action Group Name

Logic: if the agent in the zone area of which the id is 1, perform one of “cheer” actions group

Active	Priority	Logic	Not	ID	Input	Inf	Min	Inf	Max
<input checked="" type="checkbox"/>	0	&&	<input type="checkbox"/>	A	zone[1].in	<input type="checkbox"/>	1.0000	<input checked="" type="checkbox"/>	+ infinity
<input type="checkbox"/>	0		<input type="checkbox"/>			<input type="checkbox"/>	0	<input type="checkbox"/>	0

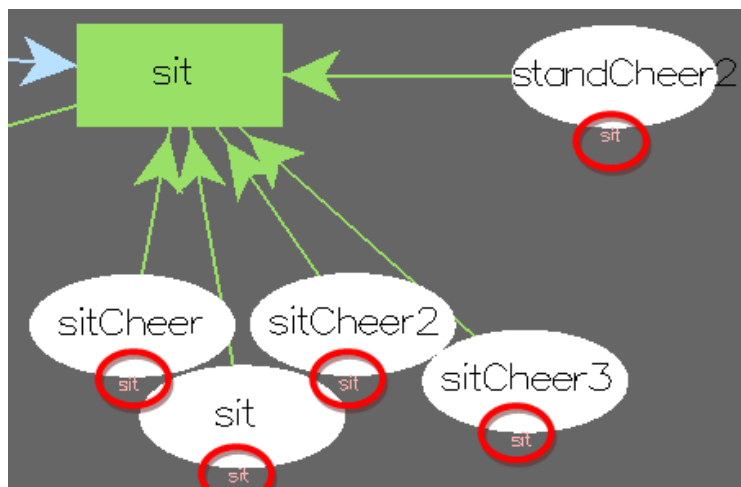
Parse Result: (A)

Active	Decision	Value
<input checked="" type="checkbox"/>	actionGroup:cheer	1.0000

Auto Fill

Trigger action by action group name

From action group, system can select one of the actions from the group. And make that selected action trigger action. The rule of select is randomization.



Randomize select one action from action group

Action Modifier

You can modify the playing back action by action blend and action rate which you specified in output channel.

Modify Action by Blending

If the current playback action has blend actions, the blended action will be performed automatically, based on the defuzzed active result of blend channel. The default blend weight is 0.0. Please notice the following things.

- Blend actions should be the same length as the main action. For example, the length of walkHappy and walkSad should be the same length as the walk action
- Blend actions can be different playback rate from main action. For example you can make walkHappy faster and walkSad slower in action editor. Our action blending engine can blend the rate of them.
- Action blend can be more than 1. That is means you can blend walkHappy, walk, walkSad together.

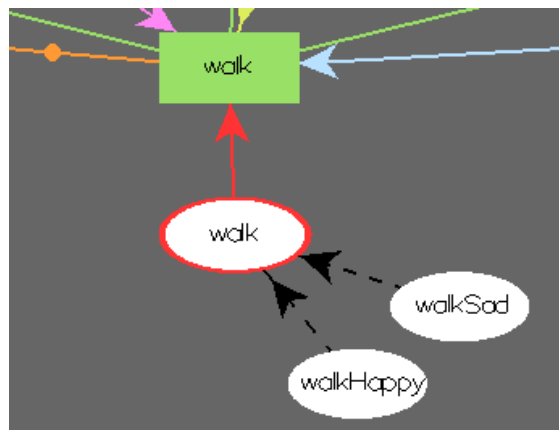
Active	Priority	Logic	Not	ID	Input	Inf	Min	Inf	Max	FuzzyIn	FuzzyOut
<input checked="" type="checkbox"/>	0	&&	<input type="checkbox"/>	A	spot.d	<input type="checkbox"/>	0.0000	<input type="checkbox"/>	100.0000	0.0000	50.0000
<input type="checkbox"/>	0		<input type="checkbox"/>			<input type="checkbox"/>	0	<input type="checkbox"/>	0	0	0

Parse Result: (A)

Active	Decision	Value
<input checked="" type="checkbox"/>	walk->walkHappy	1.0000

Auto Fill

The closer the the spot, the more walkHappy action will blend



Modify the action by blends

Modify Action by Rate

Also you can modify the speed of the playing back action by action rate channel, also based on the defuzzed active result of rate channel. The default rate is the rate in action editor.

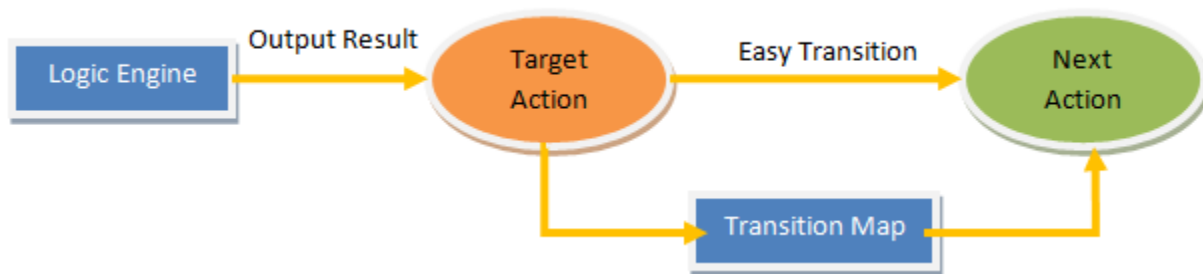
Active	Priority	Logic	Not	ID	Input	Inf	Min	Inf	Max
<input checked="" type="checkbox"/>	0	&&	<input type="checkbox"/>	A	zone[2].in	<input type="checkbox"/>	1.0000	<input checked="" type="checkbox"/>	+ infinity
<input type="checkbox"/>	0		<input type="checkbox"/>			<input type="checkbox"/>	0	<input type="checkbox"/>	0
Parse Result: (A)									
Active	Decision	Value							
<input checked="" type="checkbox"/>	jog	1.0000		Auto Fill					
<input checked="" type="checkbox"/>	jog:rate	0.5000		Auto Fill					

Modify the action by rate

You can increase the rate when the agent is walking down fast and decrease the rate when the agent is climbing.

Action Drive Agent

Pipeline



The process which we find next action

Recalling the previously part, from output results of logic calculation we can get some action outputs. And finally we find the biggest activated one, we call this **target action**. Take a look at the above picture. Then, from the target action, we will find out the **next action** by the action search method.

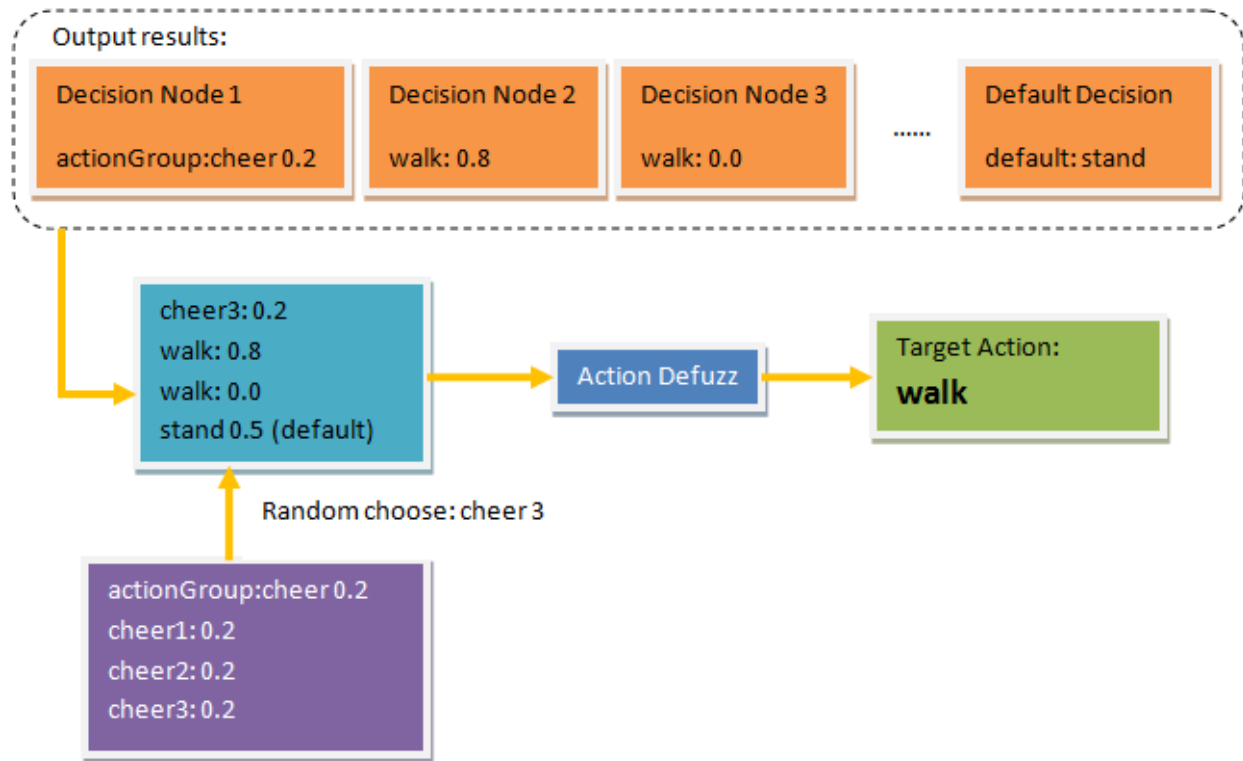
Once we solve out the next action, this is the action we need transit into. If the transition condition meets, the agent will transit in to it.

Note: the next action might be the currently playing back action itself.

Getting the Target Action

As far as we know, after logic engine calculation, we can get a list of output results. It contains many repeated channel outputs and output values. Then we need use the defuzz algorithm and the default decision node to get the behavior result.

For example, after input channel engine calculation, we get some decision results from many decision nodes, as the following first picture show.



The process of getting target action

Parsing out the Next Action

From the target action, we need solve out the **Next Action** firstly. There are several methods to solve out next action

By Simple Transition

In Simple Transition Mode (Setup in Miarmy Global), if your current playing back action is “stand”, and the target action solved out from logic engine is “walk”, the next action will be “walk” directly, because our engine can skip the transition map and transit from current action to the target action simply and directly.

By Transition Map

If the simple transition mode has been disabled, means we are in transition map mode. Suppose the current playing back action is “stand”, and the target action is “walk”, our system can automatically find the nearest action in Transition Map if you setup them correctly. The search result action will be “standToWalk” in the following example.



Transit from stand to walk with transition map

By Transition Map and Action Exits

There may be several actions in next action results, for example, take a look at the picture below. If our agent is transiting from walk to stand, our system can find 2 next actions “walkToStandL” and “walkToStandR” from transition map. By default, the system will choose the first action to transit in.

Exit Choices:					
Exit Action:	walkToStandL	Start Frame:	5.0000	End Frame:	10.0000
		Preview:	0	exit	27
Exit Action:	walkToStandR	Start Frame:	18.0000	End Frame:	22.0000
		Preview:	0	exit	27

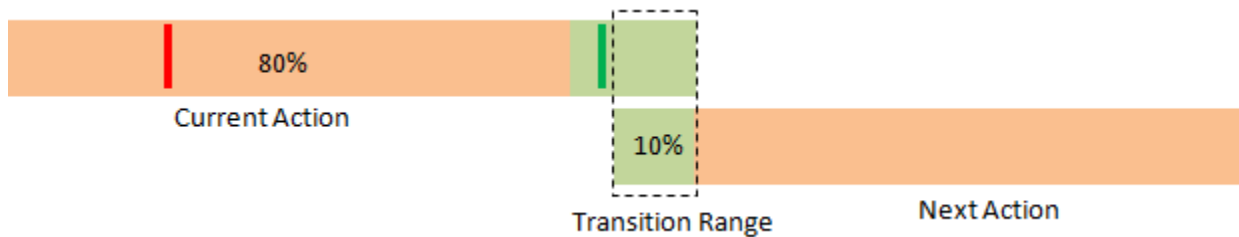
(Upper) Search engine can find 2 nearest next actions from transition map; (Lower) exits in action editor

In action editor, if we specified for “walk” action 2 exit actions, then depending on the phase of current playing back, our system can choose the exact one next action.

Like the above 2 pictures, if current frame in “walk” action is 5 to 10, the next action will be walkToStandL, whereas if the current frame in “walk” action is 18 to 22, the next action will be walkToStandR.

Action Transition

After action searching, we can find the **next action**. Then we will determine whether we need our agent transit to next action or still playback current action continually. If the current playback is before “exit range” (red cursor), the action will continue playback itself, whereas the current playback is exceed the “exit range” (green cursor), and also the next action is not itself, the transition will begin.



Transition from current action to next action

After transition, the next action will be the current playback action and store in agent memory.

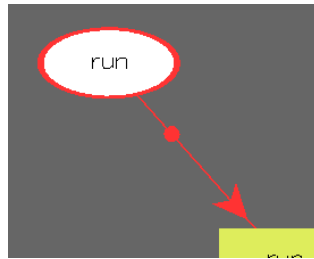
Please notice this process is automatic, here we just explained what happened under the hood.

Action Transition Visualization

Except logic feedback, our engine is able to feedback action playback status to Transition Map. The feedback is based on the marked agent, for marking an agent, just select this agent, as same as the Brain Viewer. Also see **Part 6 Logic & Perception – Feedback in Essential**

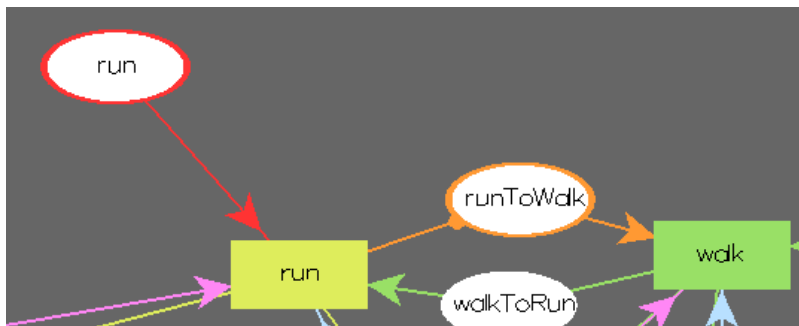
The pictures below show the process of transition. The red or orange dots are the phase indicator for the playing back actions. There are 3 types of status in transition process.

Cyclical playback status:



Cyclically playback the “run” action

Transition in process status:



Transition in process, from “run” to the “runToWalk” action

Non-Cyclical playback status:



Playing back the “runToWalk” action, non-cyclically

Go into cyclical playback status again after transition:



Transition complete, cyclical playback “walk”

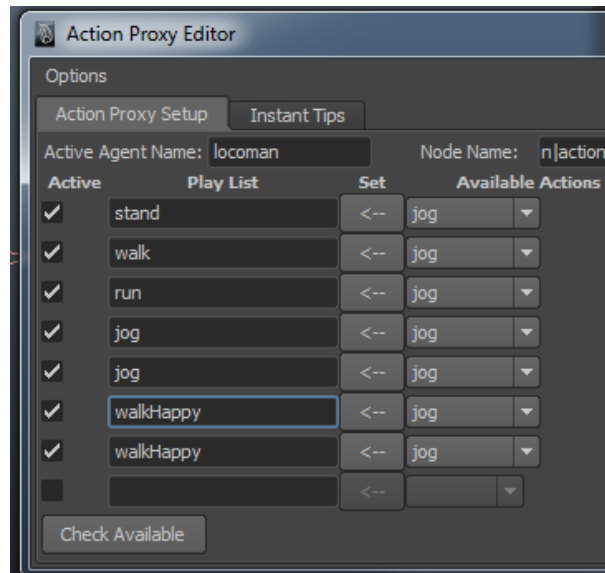
You can also disable this in for Miarmy > Miarmy Global > Stop Feedback. For details, **Part 6 Logic & Perception – Feedback in Essential**

Action Proxy List

Action Proxy List is a node contains a list of action names. The action name can be any type of actions, cycle action, transition action or blend action. This list is able to override brain logic and play actions in itself as your arrangement.

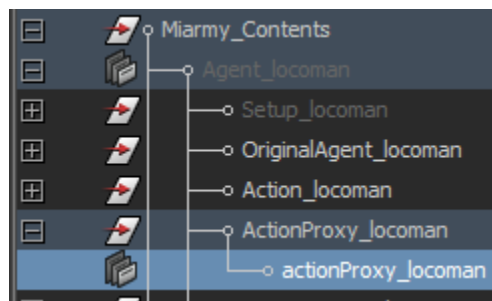
It means if you create and enable the action proxy, any of logic will be ignored. The agent will perform the actions in list sequentially just like pre-defined rehearsal.

We usually can use the action proxy list to test action continuity.



Action Proxy Editor

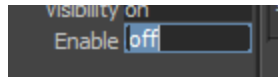
Action Proxy Node always located in ActionProxy_<agentName> group, and only one action proxy node can be created for each one type of agent.



Action proxy node

And you can edit it by Action Proxy Editor.

If you want to disable the proxy list without delete it, please turn off the “enable” attribute of it.



Disable without delete action proxy

Note: in this version of Miarmy, in action proxy list, if the next action is the same as current one, they will transit by “entry range” and “exit range” mechanism instead of “cycle range”. We will fix this in next minor upgrade.

Part 7 Physics

From design, we planned integrate NVIDIA PhysX engine into Miarmy. Miarmy now can easily handle more than 5000 agents and more than 100,000 rigid bodies on a Windows HOME PC. As our design, all the physical contents are calculated directly in PhysX engine and we just display them in Maya, so, there is no any bottle neck between PhysX and Miarmy when simulation in process.

Enable Dynamics

Dynamics Types

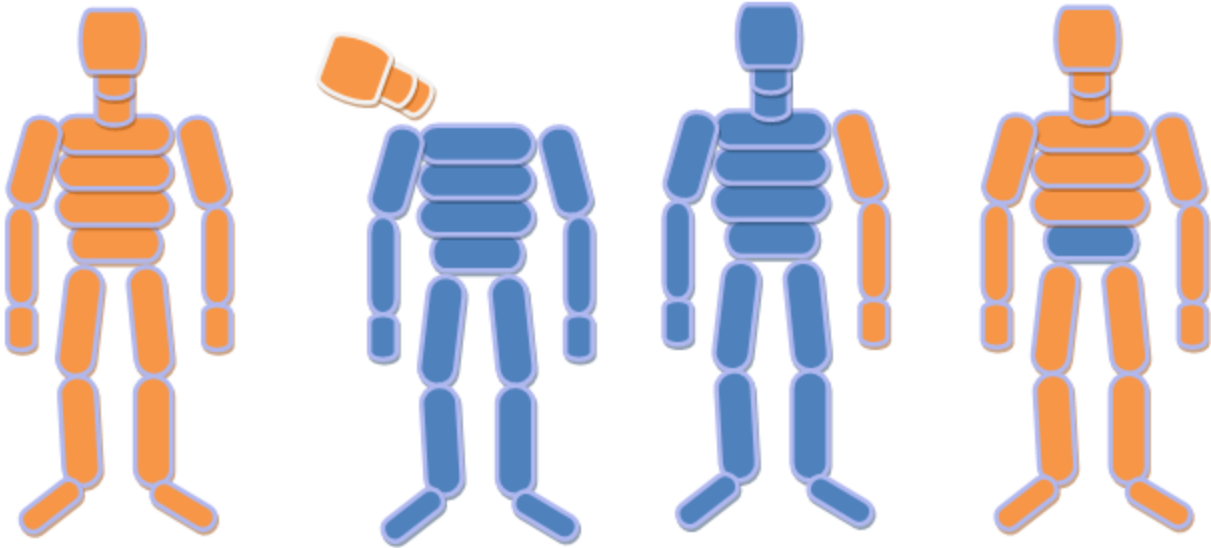
You can easily enable agent ragdoll-like dynamics by decision channels. Basically, there are 4 types of dynamics channels, they are:

1. **Full body dynamics**: enable dynamics for every part of agent
2. **Detach dynamics**: break a bone and its sub tree bones from agent, but keep the rest of agent controlled by animation.
3. **Partial dynamics**: enable the sub tree dynamics from a bone, but keep the rest of agent controlled by animation.
4. **Body dynamics**: enable dynamics for every part of agent except the root bone.

The illustration of each type of dynamics, the blue block are the bones controlled by action whereas the orange blocks are controlled by dynamics.



Normal status



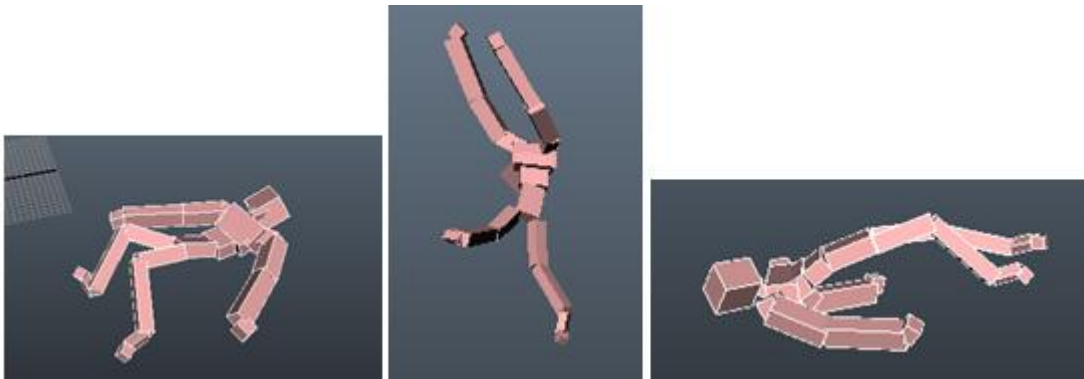
(1) Full body dynamics (2) Detach dynamics, (3) Partial dynamics (4) Body dynamics

Note: in this version of Miarmy, the (3) Partial Dynamics and (4) Body Dynamics have some problems due to the Bugs from PhysX engine. The NVIDIA is fixing it and we believe that they can fix it soon, and we will update this part as soon as they fix it. For details please refer the following link:

<http://forums.developer.nvidia.com/devforum/discussion/3191/bug-physx-3-1-joint-strange-movement-video>

Dynamics Channels

The channel “**dynamics.active**” can enable full body dynamics for the agent.

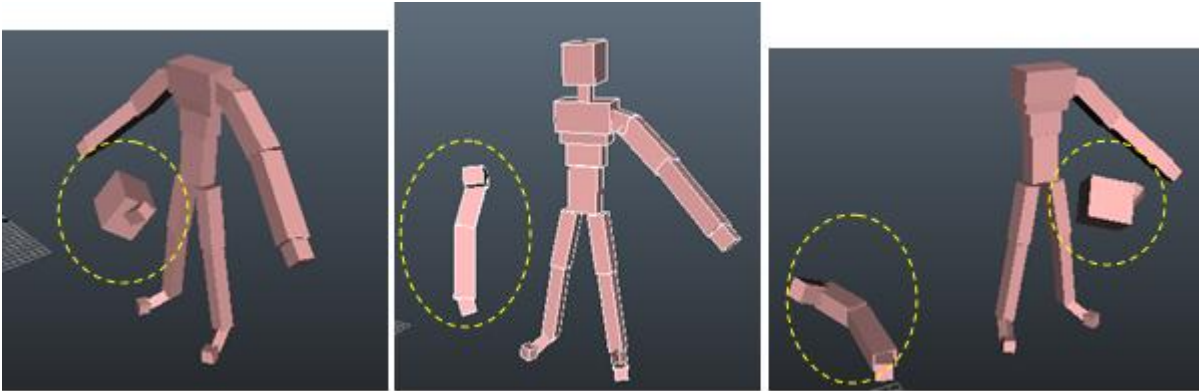


Full body dynamics

The Channel “<boneName>:dynamics.detach” can enable partial detach dynamics for agent. The dynamics will be enables from the specific bone and its sub tree, and break from that bone. This is easily simulate the zombie like character walking and being shot.

Example:

- **neck1:dynamics.detach**
- **shoulderR:dynamics.detach**

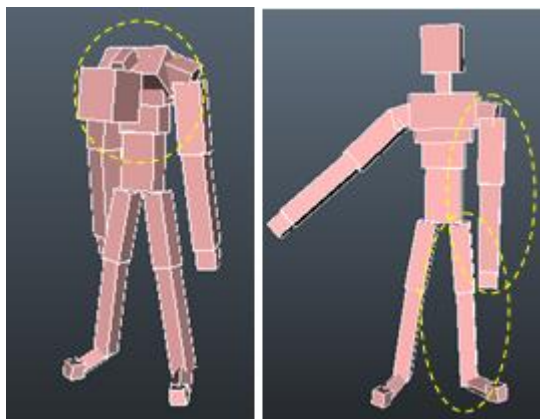


Detach dynamics (head, arm, head + arm)

The channel “<boneName>:dynamics.active” can enable partial dynamics from the specific bone and its sub tree, but not detach from main agent. The main agent will be still controlled by action and Logic

Example:

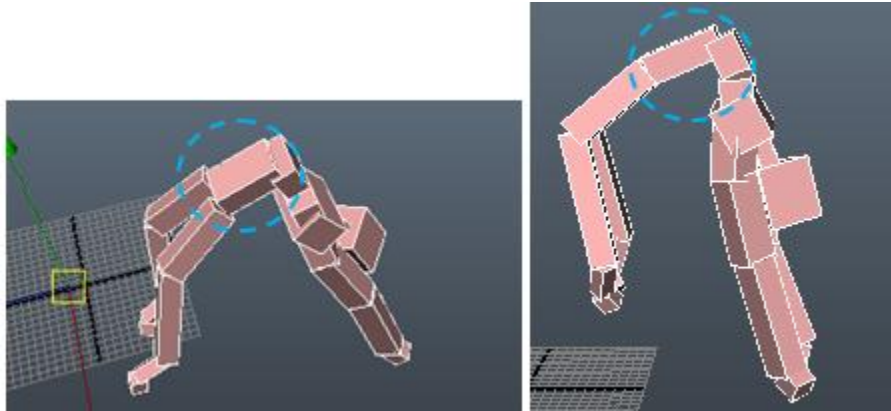
- **torse:dynamics.active**
- **shoulderL:dynamics.active (upLegL:dynamics.active)**



Enable partial dynamic from torso3 and shoulderL

The channel “<bodyDynamics.active” can enable every bone of agent except root bone. And you can control the root by animation and the rest of bones will perform dynamic simulation. Just like you are waving a Nunchaku,

the handler of it is controlled by animation and the rest are controlled by dynamics. You can constraint each of agent to a particle and write some expression to control them.



Body dynamics enable except root bone

Collision Detection

Collide Check Channels

Using collide channels, we can detect the bone collision status of agents.

For getting the details how the collision detection work and optimizing it with mute dynamic, please refer **Part 7: Physical Simulation – Collision Detection**

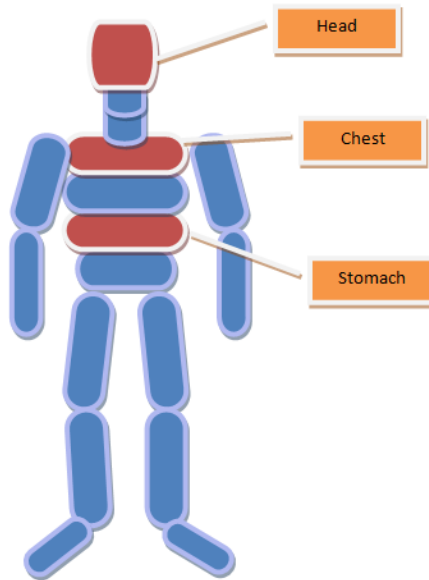
Note: The bone will not check collide with terrain, only interactive with them. And the bone also can check collide with kinematic primitive objects, which will be explained in **Part 7: Physical Simulation – Kinematic Primitives**

Using **collide** channel we can check the collision status for the bones which marked “feel collide” from the current agent. If you didn’t remember what is bone which marked “feel collide” flag, please refer **Part 3 Agent Infrastructure – bone flags**

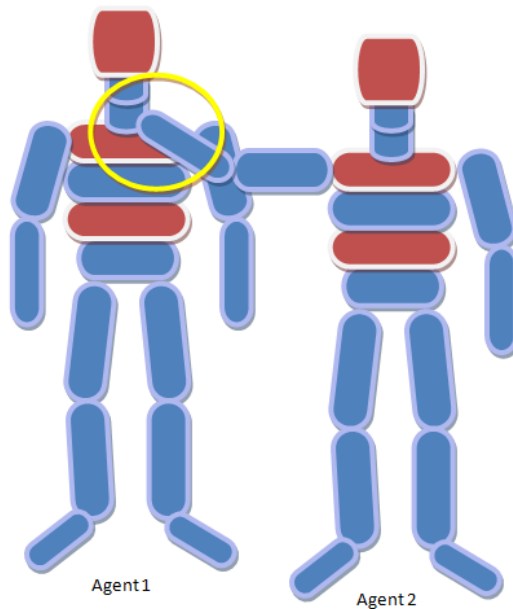
If the bones which marked “feel collide” are colliding/overlapping the others RBD, the channel will return 1, otherwise, return 0. Just like the picture below, the “**head**”, “**chest**”, and “**stomach**” bones have been marked “feel collide” flag,

For the agent 1, the “**collide**” channel will return 1 because the “chest” is colliding with the hand of agent 2, whereas,

For the agent 2, the “**collide**” channel will return 0 because the bones marked “feel collide” is not colliding with any others bones.



Head, chest and stomach have been marked “feel collide” flag



collide channel will return 1 from agent 1 and return 0 from agent 2

collideAny channel will return 1 from both agent 1 and agent 2

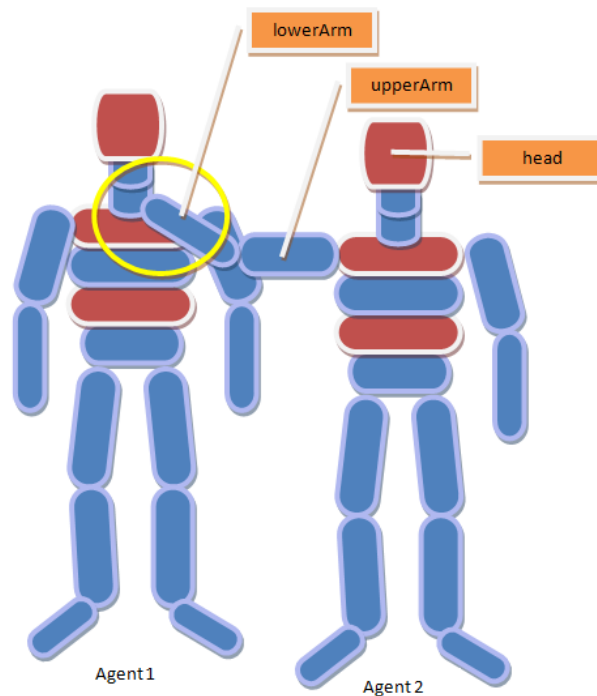
Using **collideAny** channel we can check the bone collision for all the bones of the current agent. Take a look at the picture above, the agent 1 and agent 2 collided, so no matter which part are they colliding, the channel will return 1 for both agents.

Note: the “collideAny” is much slower than “collide” because it will check collide for any bone of current agent. Just like the current example, there are totally 17 bones of each agent and 3 of 17 are marked “feel collide”. So the time spends on the “collide” channel is only 3/17 (0.177) of time spend on “collideAny”. It’s 5.6 times faster. Imagine that you have 100 bones and only 1 marked “feel collide”, and the “collide” channel is actually 100 times faster than “collideAny” channel.

Using “**collideBy:XXX**”, we can check is there a bone which name is XXX collide with the bones marked “feel collide” of current agent, for example:

For the agent 1, **collideBy:lowerArm** will return 1 because the chest bone of current agent is colliding with a bone which name is “lowerArm”

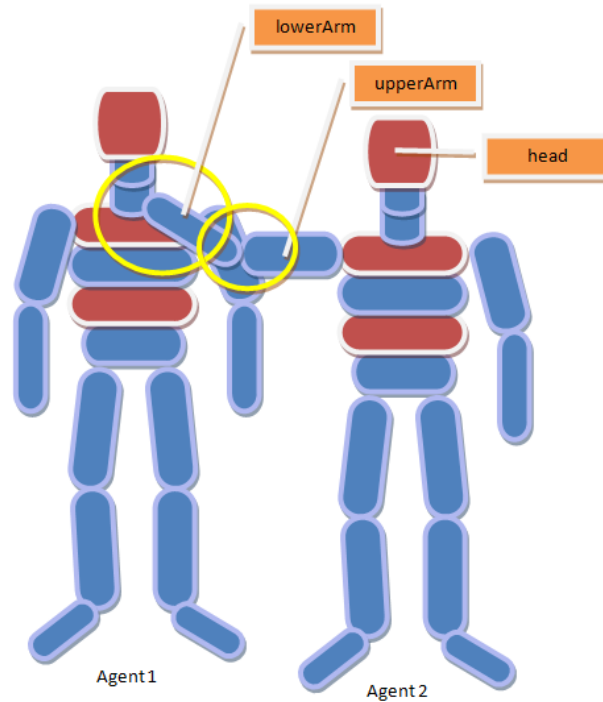
For the agent 1, **collideBy:head** will return 0 because no matter “chest”, “stomach” or “head” is not colliding with the bone which name is head



collideBy:lowerArm will return 1 and collideBy:head will return 0

Using “**collideBy:XXX**”, we can check is there a bone which name is XXX collide with the any bones of current agent, for example:

For the agent 1, **collideAnyBy:upperArm** will return 1 because the upper arm of agent 2 collide with one of the bone of agent 1



collideAnyBy:upperArm will return 1

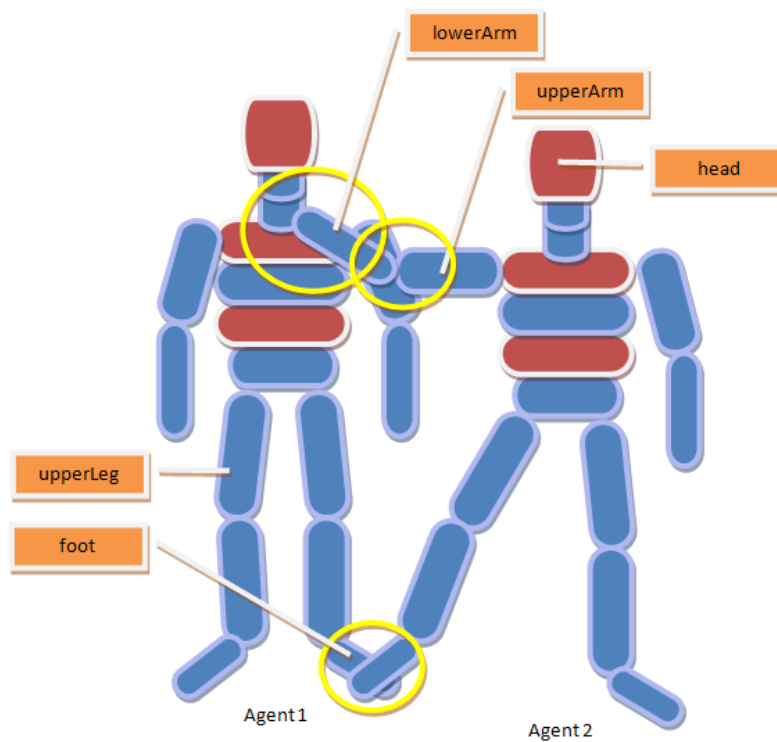
Using “**collideAt:XXX**”, we can check whether the bone name is XXX from current agent colliding with any else bone from others agents. No matter the bone XXX whether or not marked “feel collide” flag.

Look at the following example:

For agent 1, the “**collideAt:chest**” will return 1 because the chest is colliding with a bone

For agent 1, the “**collideAt:foot**” will return 1 because the foot is colliding with a bone

For agent 1, the “**collideAt:upperLeg**” will return 0 because the upper leg is not colliding with any bone

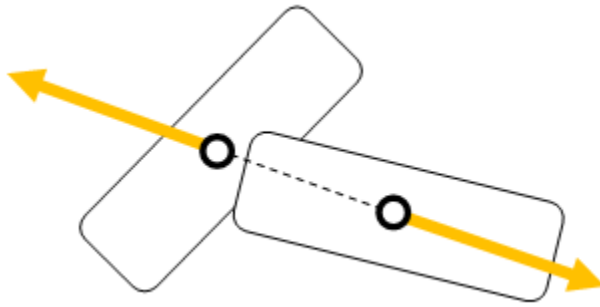


collideAt:foot will return 1, whereas collideAt:upperLeg will return

The “ri” prefix is use to reserve information of collide. The information will contain the collide direction and velocity.

- [riCollide](#)
- [riCollideAny](#)
- [riCollideBy:XXX](#)
- [riCollideAnyBy:XXX](#)
- [riCollideAt:XXX](#)

The reserved information contain the approximated collision **direction** and the **relative velocity**



The collision info reserved when checking collide

We can apply a pulse force on to it when enable dynamics, with the channel “[dynamics.active.force](#)”. The force direction will along with the reserved vector and the force strength will be the defuzzed value of the channel.

Pre-build RBD Objects

Before us diving into the others dynamics features, we need deeply understand a very important concept. That is pre-build RBD objects

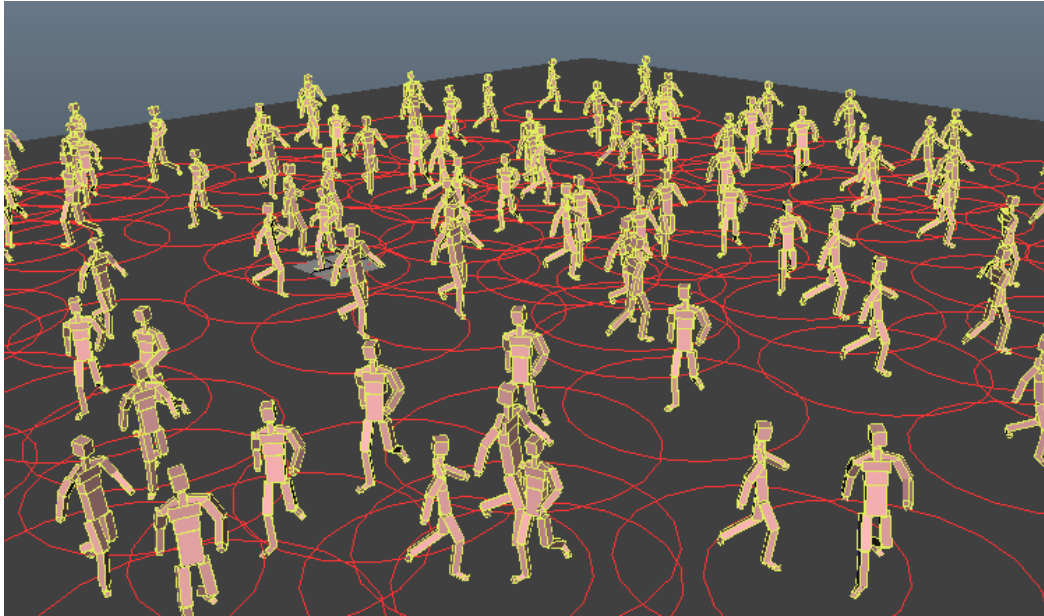
In some case, we need create some pre-build the RBD objects in agent memory and let out animation drive these RBD before really physical simulation. In this time, you cannot see agent perform physical simulation, but actually the physical contents are there and controlled by action.

Firstly, let’s take a look at that **when** these pre-build RBD objects will be created and **why** we need them, and **how** can we know/detect their existence. Then we will introduce you **how** can you optimize and avoid our engine create these pre-build RBD objects if our scene doesn’t need them.

Pre-build RBD Objects for Collision Detection

We are using the PhysX feature for checking collision, so, if there is “collide” related channels in input sentence, we need pre-build RBD object in agent memory. These RBD objects are controlled by action and the user may not notice them. Our system will use these RBD objects for collision checking and returning check result.

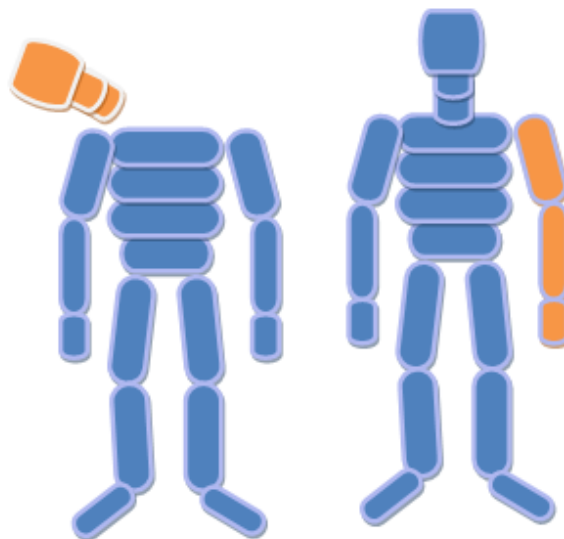
You can visualize check these RBD objects by debug tool: Miarmy > Debug Tools > Create PhysX Debug Node. As the following picture shown, the agents have “collide” checking channel. The yellow contour outside the agent means that the agent have RBD object inside the memory, even they are driven by action.



The agent with RBD pre-build in memory

Pre-build RBD Objects for Dynamical + Action Hybrid Drive

Once we enable partial dynamics or break dynamics. Except the dynamic parts, the rest of agent bones have pre-build RBD object there in agent memory. For example the following picture, the orange parts have been enable dynamics but the blue parts are driven by action. However, the blue parts also have RBD object there in agent memory



The blue parts also have inner RBD objects in memory

Pre-build RBD Objects for Cloth Collide

Apparently, we have to create pre-build RBD objects for collide with cloth when there is cloth on agent.

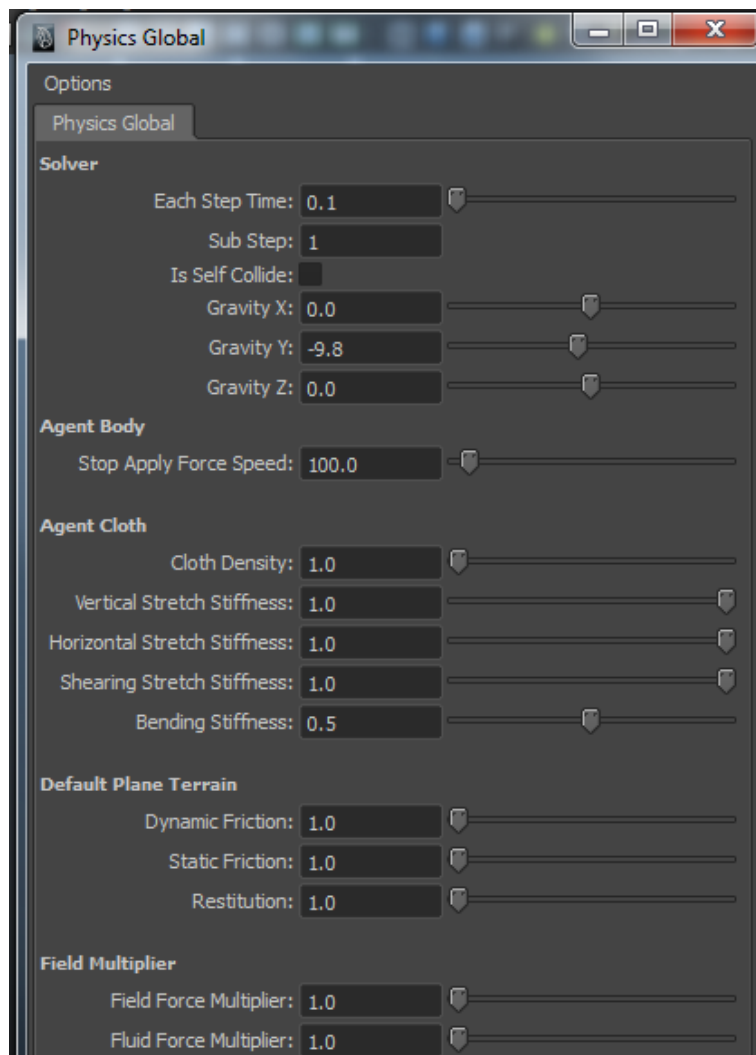
Optimization

The pre-build RBD will take huge memory if you have huge number of agents in scene. See **Part 9 Optimization – Mute Dynamics**

Dynamics Features

Physics Global

Miarmy > Physics Global



Step Time and Sub Step

Step time is the time span of physical simulation in a single Maya frame. By default, in each Maya frame, the PhysX simulate and increase 0.1 second for updating physical contents.

Sub step is the number of steps in each simulation time span. For example, if sub step is 4, and step time is 0.1, each time PhysX simulate, it will simulate 4 times and each time takes 0.025 second.

Increase sub step can achieve precise simulation result but take more time to calculate.

Let's take a look an example, when the agents are pushed by force field, the different sub step lead to different precise level.



Sub step from left to right: 1, 2, 4

Gravity VS Mass

You may encounter these situations:

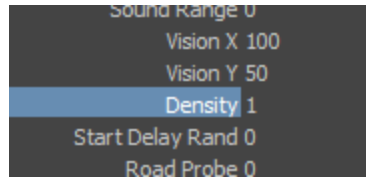
- The object from high place falling slow
- Explosive effect not much

If the agent from high place falls to ground slowly, it means the gravity not enough. At this time you need increase the gravity. On the other hand, if you increase the mass of agent body, there will be no any change. You can just recall the famous story about Galileo's 2 balls.



Increase the gravity

If the force field like explosive force effect agent not much, mean the object is too heavy. At this time, you need crank down the density for decreasing the mass of the agent bones. $\text{Mass} = \text{volume} * \text{density}$, the volume is the bone shape itself.



Decrease the mass

Galileo, he is probably best known for a story in which he dropped two different size balls from the Leaning Tower of Pisa at the same time, and the two balls hit the ground at almost the same time



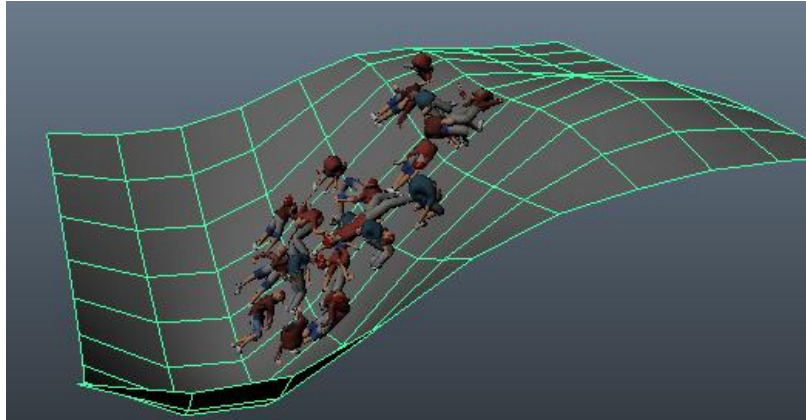
(Left) Leaning Tower of Pisa, (right) Galileo

Terrains and Default Terrain

In the previous **Part 5 Logic & Perception**, we talked about the terrain interactive with agent by logic channel. In addition, the marked terrain can interactive with dynamical agents, automatically. Dynamical terrain can be any shape, even deforming when simulation.

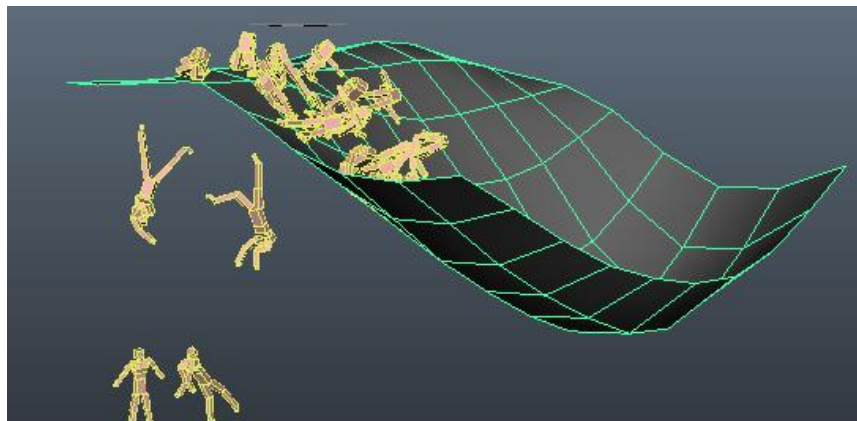
The only thing need to do is mark your terrain in Terrain Manager. Please notice:

- If the terrain is deforming or moving when simulation, please mark the “isAnim” flag in Terrain Manager. Our system can update the shape of it every frame.
- If the terrain is a plane, you can mark the “isPlane” flag for accelerate the calculation.



Dynamical agents interactive with terrain

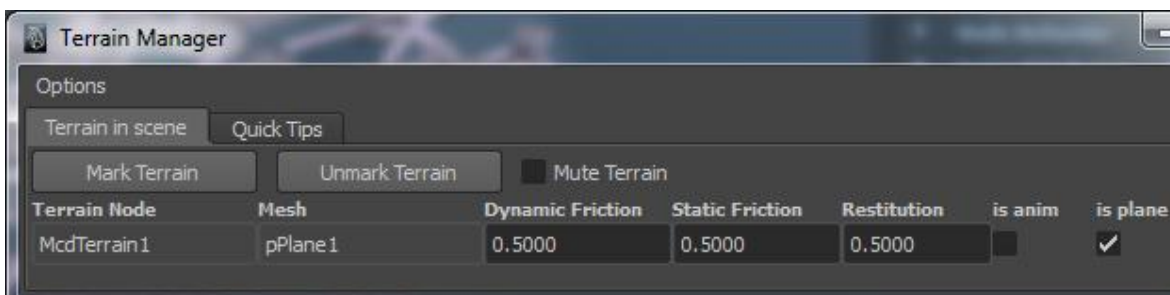
Please notice the geometrical terrain has edge. If the agent position exceeds the edge range, the agent will fall down to the abyss.



Fall down to abyss if outside the terrain range

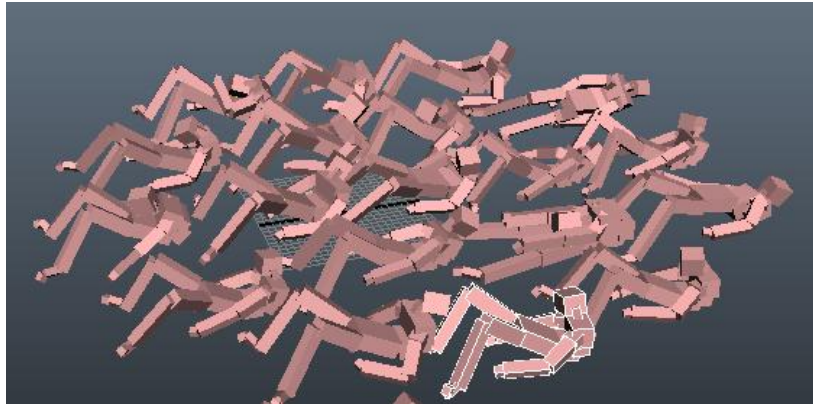
There are some attributes on the terrain:

- **Dynamic Friction:** friction when object moving on terrain
- **Static Friction:** friction want to prevent object move on terrain
- **Restitution:** you can consider it bounce capability

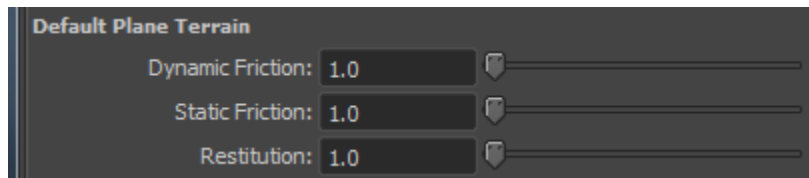


Terrain attributes

If there is no marked terrain in scene, our system will build a default terrain. The default terrain is actually an infinite ground plane primitive. And its attributes are located in Physics Global.

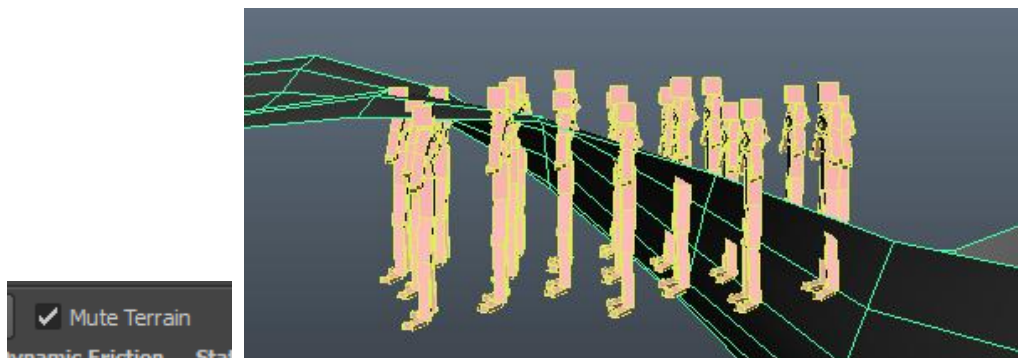


Default terrain plane



Attributes of default terrain plane in Physics Global

If you intentionally don't want any terrain in scene, you need check on "mute terrain" in Terrain Manager, then you scene will never create any terrain any more. And the existed terrain will be ignored.

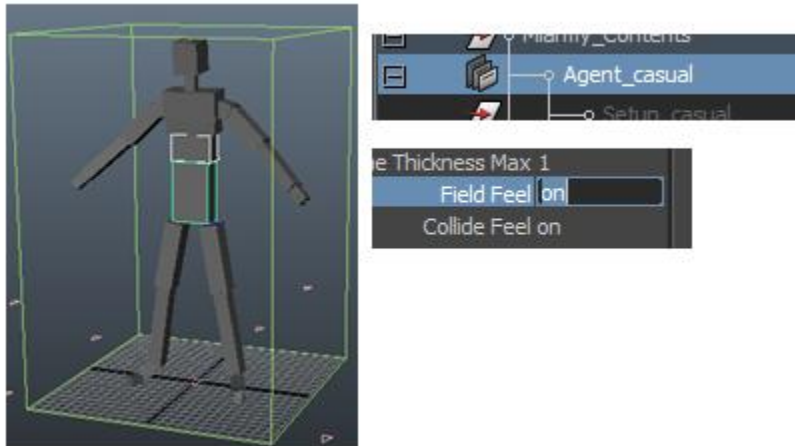


Ignore exist terrain when mute terrain check on

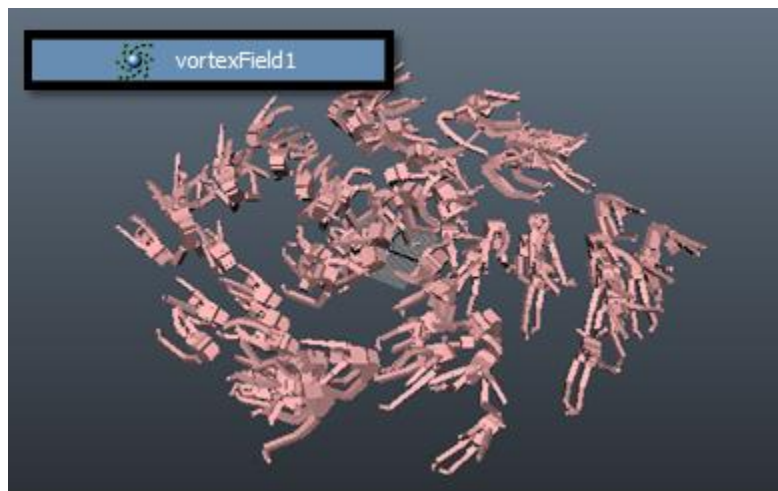
You can use check collision between agents and kinematic primitives using channel "[collideBy: _TERRAIN_](#)".

Maya Field

Miarmy agent can interactive with Maya field not only from logic input channels, but also directly in dynamic simulation. When the agent dynamics enable, the agents can directly affect by Maya field. But please notice the field can only affect the bone which marked “fieldFeel”. You need setup the “fieldFeel” flag on the bone shape of original agent. For reviewing the original agent and agent type, please refer **Part 3 Agent Infrastructure**



Mark “fieldFeel” on bones of original agent



Maya vortex field affecting dynamical agents

For fine tuning the dynamical result, you need just change the attributes in Maya field node itself.

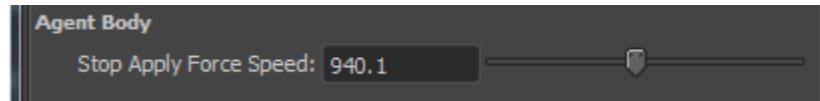
Force Field General Rules

Force Composition

No matter currently we are talking about Maya Field, or further we are going to introduce Maya Fluid, and build-in force field, all of them drive agents by **Force Composition**. For example there are several Maya fields in scene, and our agents can feel the composition force result of them.

Speed Threshold

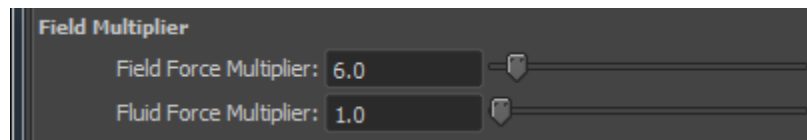
And, no matter Maya fluid, field, build-in force field, they will apply force to the RBD object only when the speed of this RBD slower than the speed threshold. Another words, once the RBD object exceed the speed threshold, our system will not apply force onto it any more.



Speed threshold value in Physics Global

Force Multipliers

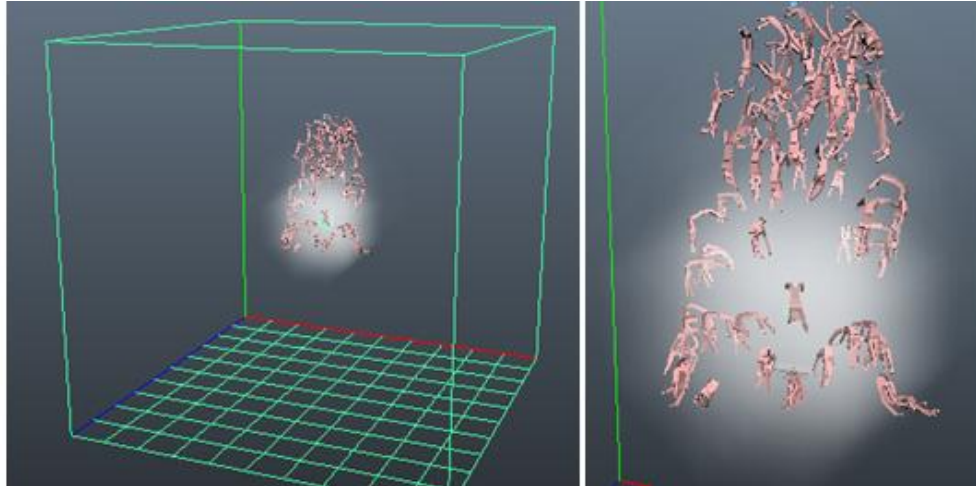
Sometimes, maybe your Maya field not only driving Miarmy Agents, but also controlling the particles. At this time, change the magnitude of Maya field will effect both on agents and particles. However, maybe the particle result is OK you don't need change it. At this time we provide the Field Multiplier for exclusively change the result on the agents. If the magnitude of a field is 100, and the field force multiplier is 6, the force applied to the agents will be $100 * 6 = 600$ from this field.



Field Multiplier in Physical Global

Maya Fluid

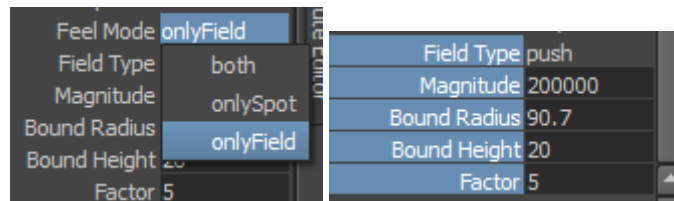
Because Maya fluid can be represented by field, the Miarmy agent can interactive with Maya fluid directly also when the agent dynamics enable. The process is the same as Maya Field.



Maya basic fluid affecting dynamical agents

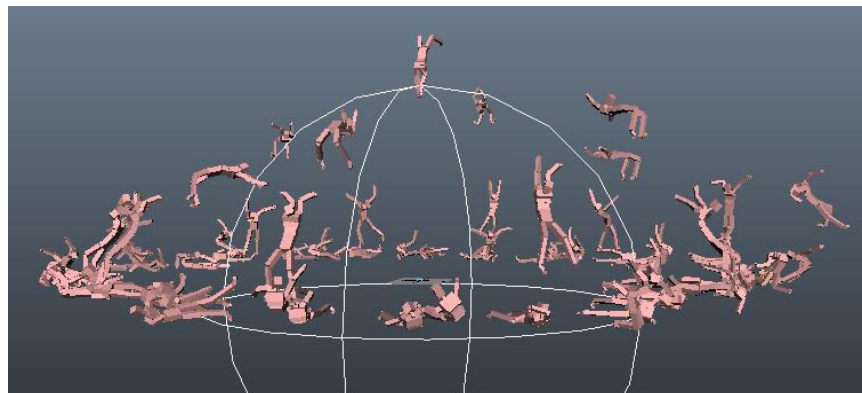
Build-in Force Field

In the previously chapter, we talked about the spot node, it's a kind of perception contents. Actually, it can be a force field as well. When you change the Feel Mode attribute to “onlyField” mode, the spot will become a build-in force field.

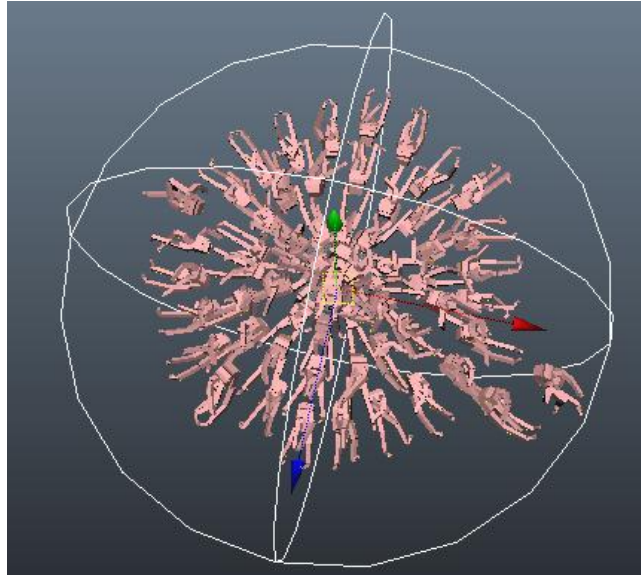


(left) Feel mode change to “onlyField” (right) the force field specific attributes

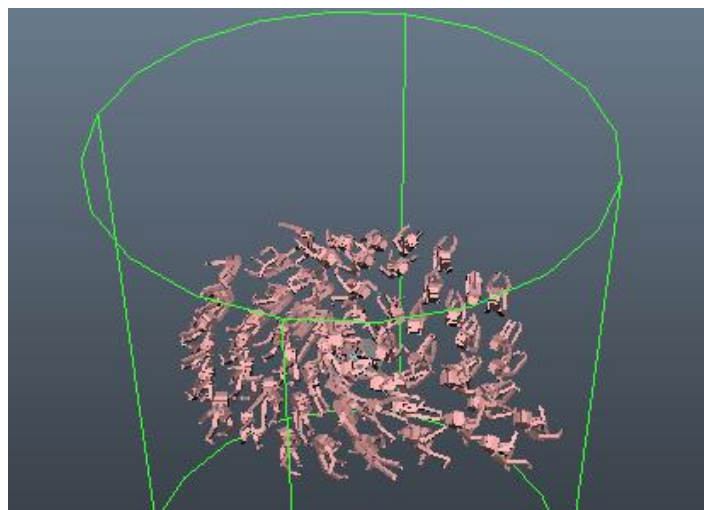
If the agent enable dynamics and some of bones marked “fieldFeel” enable, the agent will be affected apply force inside of the bound.



Push field



Pull field



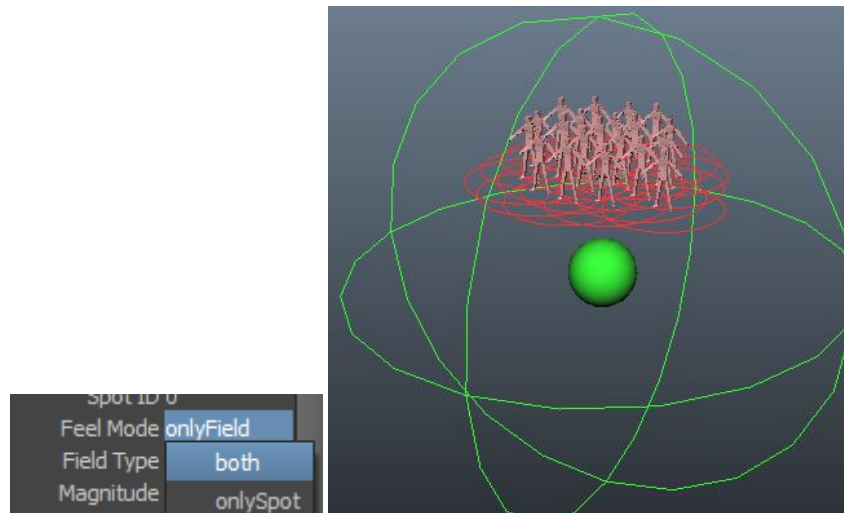
Vortex field

Build-in force field attributes:

- **Field Type:** can be push, pull and vortex
- **Bound Radius:** control the radius of bounding sphere or bounding cylinder
- **Bound Height:** (only vortex) control the height of the bounding cylinder
- **Factor:** (only vortex) the force multiplier when the agent far from center of cylinder

Combo

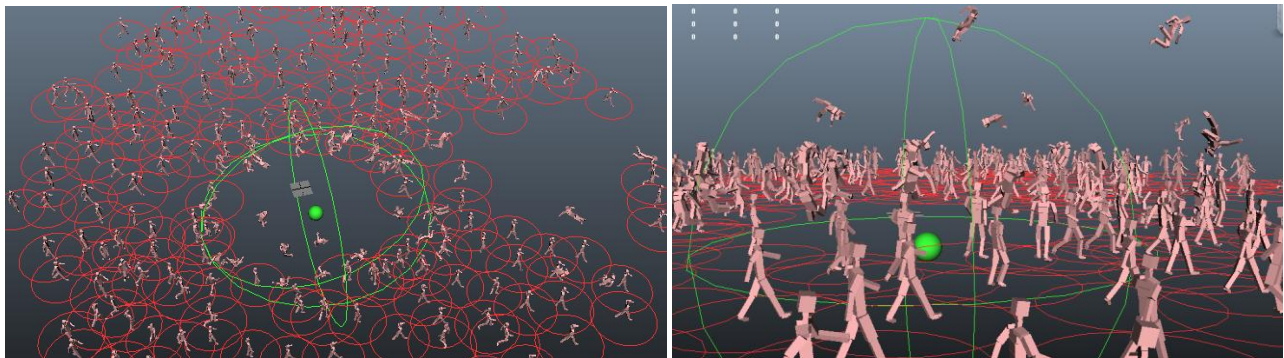
We designed spot and build-in force field together because in many cases we need them work together. Firstly we need switch the Feel Mode attribute to “both”



Combination of force field and spot

Suppose we are building an explosive scene. We can use spot logic trigger agent enable dynamics and use force field push them out form bounding sphere. At this time, the combo of spot and force field is much easy to operate.

Like the picture shown below, after using “spot.d” channel enable agents dynamics, these agents will be naturally inside the bounding sphere, and immediately they will be affect by that force field and been pushed out from the bounding sphere center.



Combo example spot trigger dynamics, force field push agent out

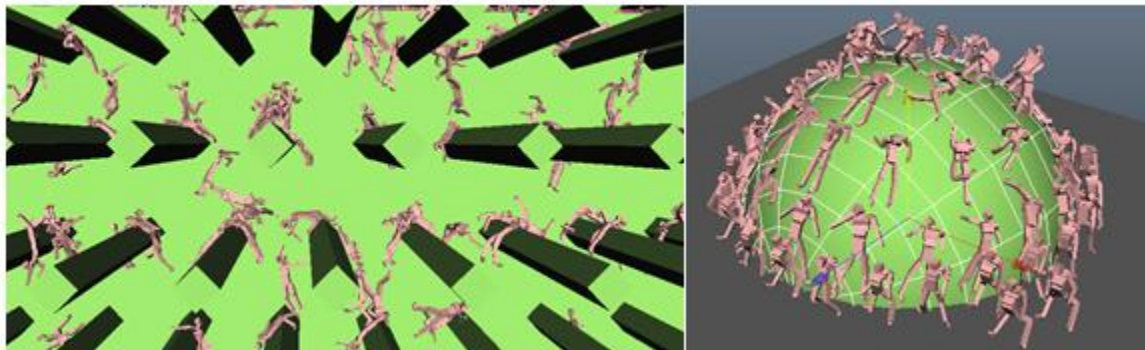
Kinematic Primitives

Kinematic primitive is a kind of static RBD objects can be controlled by user animation or constraint. Our agent can collide with them automatically if they enable dynamics.

You can scale, rotate, translate and key frame the kinematic primitive anytime. Or you can parent, constrain them to anywhere.

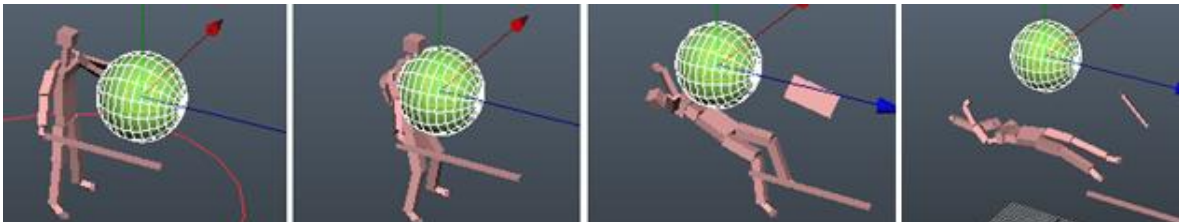
In this current version of Miarmy, only 2 types of kinematic primitives supported, they are “Box” and “Sphere”.

For creating them, just simply click Miarmy > Physics > Kinematic Primitives.



Kinematic Primitive example

You can check collision between agents and kinematic primitives using channel “**collide**” or if you want agents exclusively check collide with kinematic primitives: “**collideBy: _KINEPRIM_**”.



Check collide by “**collideBy: _KINEPRIM_**” enable dynamic + keyframe kinematic primitive

Active	Priority	Logic	Not	ID	Input	Inf	Min	Inf	Max	FuzzyIn	FuzzyOut
<input checked="" type="checkbox"/>	0	&&	<input type="checkbox"/>	A	collideBy: _KINEPRIM_	<input type="checkbox"/>	1.0000	<input checked="" type="checkbox"/>	+ infinity	0.0000	
<input type="checkbox"/>	0		<input type="checkbox"/>			<input type="checkbox"/>	0	<input type="checkbox"/>	0	0	0
Parse Result: (A)											
Active	Decision		Value								
<input checked="" type="checkbox"/>	dynamics.active		1.0000		Auto Fill						
<input checked="" type="checkbox"/>	shield:dynamics.detach		1.0000		Auto Fill						
<input checked="" type="checkbox"/>	sword:dynamics.detach		1.0000		Auto Fill						

The logic in the above example

Cloth simulation



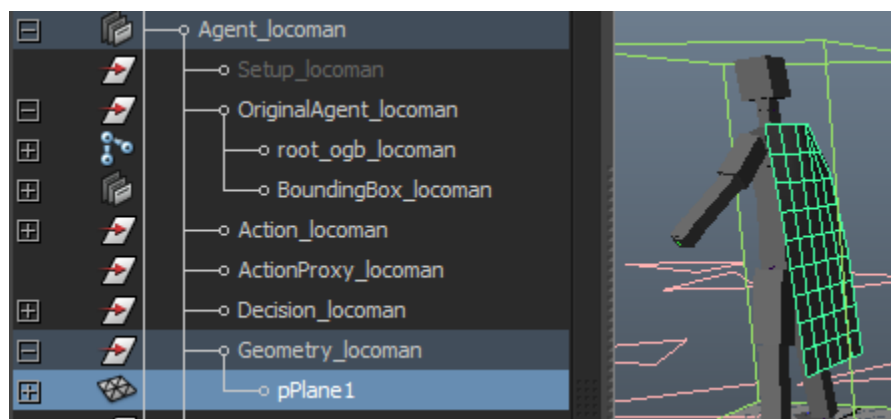
Cloth simulation

Notice: NVIDIA rewrote entire PhysX cloth system, and the latest cloth will be simulated in an independent solver space and not join the rigid body collision. It's much faster than previous version (the more clothes the faster, at least 10 times faster and more).

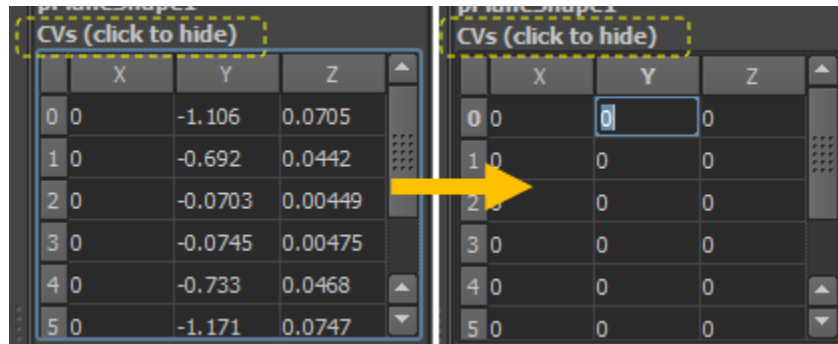
But the cloth feature is not fully been finished all by NVIDIA, such as collide with ground, cloth friction and so on. We will provide more features later in next version.

Creating Cloth

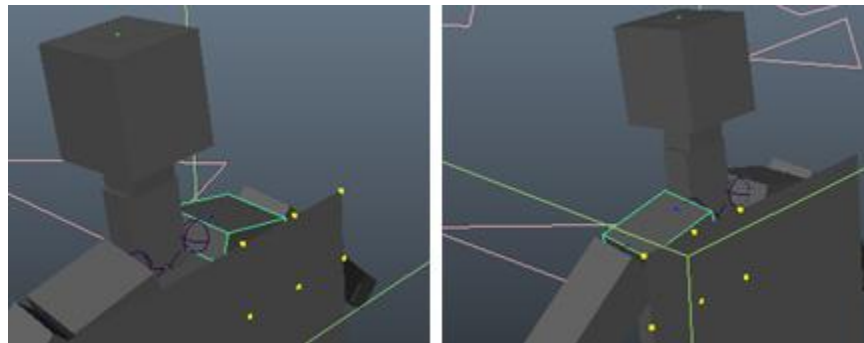
1. Put your cloth geometry directly under the Geometry_<Agent Type> group
2. Make sure your cloth do not contain tweak info
3. Select attach points and bone shape box of Original Agent, click Miarmy > Physics > Cloth Setup > Attach Cloth
4. You can continue add attach points to different bones from the single cloth



Step 1: put you cloth to Geometry_<Agent Type>



Step 2: Make sure you clear the tweak info on cloth mesh



Step 3: Attach points to bone shape box of Original Agent

Tip: if your geometry mesh has tweak info, but you don't want to change the topology (shape) of that mesh, you can simply "add a cluster deformer" to the mesh and then "delete history" from the mesh, the 2 steps will help you clear the tweak info on the mesh but will not modify the points position

Once your setup complete, and place your agents from placement node, the clothes will appear.

Cloth Attributes

There are several clothes attribute for uniformly control all clothes in Physics Global. And these attributes



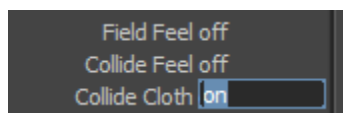
Cloth attributes in Physics Global

- **Solver Frequency**: Solver frequency specifies how often the simulation step is computed per second
- **Stiff Damping**: damping coefficient for stiffness of cloth
- **Cloth Density**: the mass for each particle on mesh
- **Vertical Stretch Stiffness**: stiffness for holding the length in vertical space
- **Horizontal Stretch Stiffness**: stiffness for holding the width in horizontal space
- **Shearing Stretch Stiffness**: stiffness for holding the shape
- **Bending Stiffness**: stiffness for resisting shape bend

Note: because the cloth feature is under a major rewrite by NVIDIA, some features are not very stable. Please use it carefully.

Cloth Collision on Bone

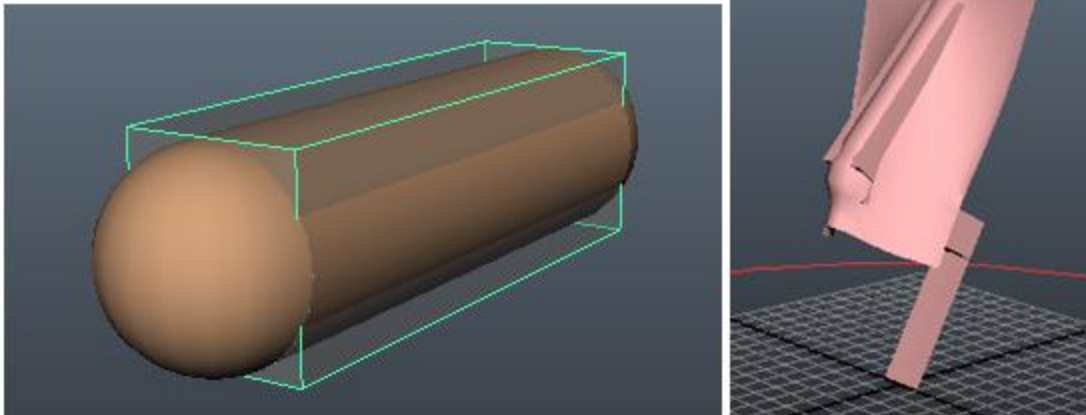
If we want to our cloth collide against with the bone, we need set the flag in original agents firstly.



Turn on collide cloth flag on bone shape

The cloth in PhysX 3 can only collide with capsule, so our system will create an approximate capsule based on the shape of cubic bone.

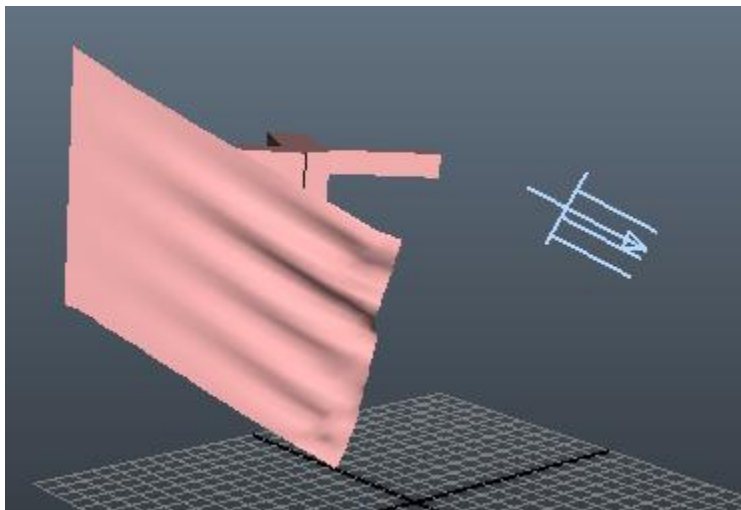
Each cubic bone has 3 dimensions which are X, Y and Z. Our system will choose the longest dimension for the capsule length and the second longest for the radius of this capsule. Just like the left example below. Then once we enable simulation, the cloth will collide against the bone, like the right picture below.



Approximate collision object from cubic bone

Cloth affected by Wind

Cloth can only affect by wind force field. The wind can be created in Miarmy > Knowledge Perception > Create Wind.



Cloth affected by wind

The wind will blow to the direction which its arrow pointing. And the magnitude will be the value between (magnitude + noise) and (magnitude – noise). In the following example, the wind magnitude result will range from 0 to 100. (50-50, 50+50)

Wind Mag 50
Wind Noise 50

The wind attributes

Mute Dynamics

There is a “muteDynamic” attribute on each agent. Once we turn this on, the agent will never enable dynamics and will not join the collision detection, the pre-build RBD object will never been created. In some cases, using this feature properly, it will save tremendously amount of memory for you as well as speed your simulation up. See some real example in **Part 9 Optimizing**.

Additionally, this attribute cannot save with scene, if the agents have been deleted and placed out again, these attribute will change back. So, you need enable “muteDynamics” first and use “inverse placement” to store these data to a new place node.

Part 8 Render

Agent Cache

Let's firstly check out the agent cache before rendering. One can easily and fast create Miarmy Agent Cache for storing the simulation data of agent bones to external files. Then, using the agent cache files, artists are able to real time drag back and forth on the time slider and get the directly visual feedback.

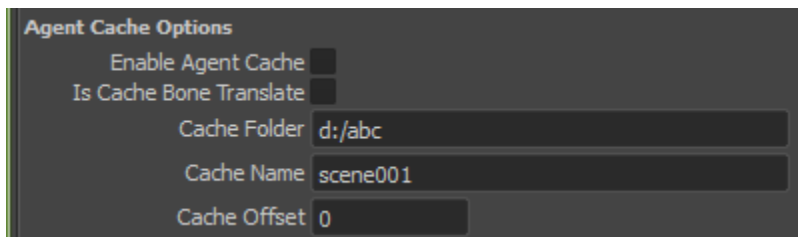
Agent cache is one file for each frame. It contains the simulation data in this frame. Usually the simulation data is the bone rotates and translates, also the cloth information if there are clothes on agents.

The extension name (usually called format) of cache file is ".mmc", and the naming convention is "<CacheName>.<Frame Number>.mmc", e.g. "scene001.26.mmc".

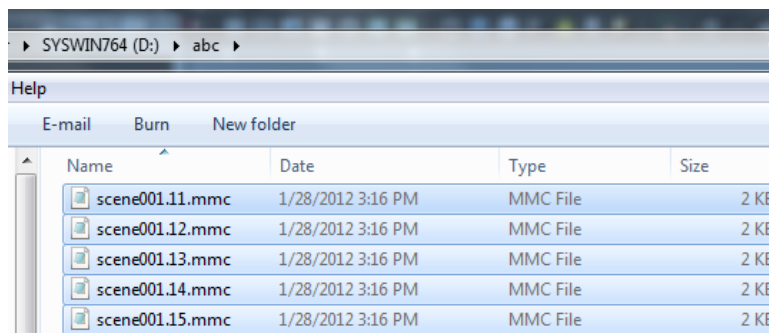
Only with render cache, we can apply motion blur render.

Creating Agent Cache

1. Make sure you fill the right agent info in Miarmy Global
 - a. Cache folder: the cache file located
 - b. Cache name: the name of cache files
2. Click Miarmy > Make Agent Cache
3. Enable Cache in Miarmy Global



Step 1: fill right information

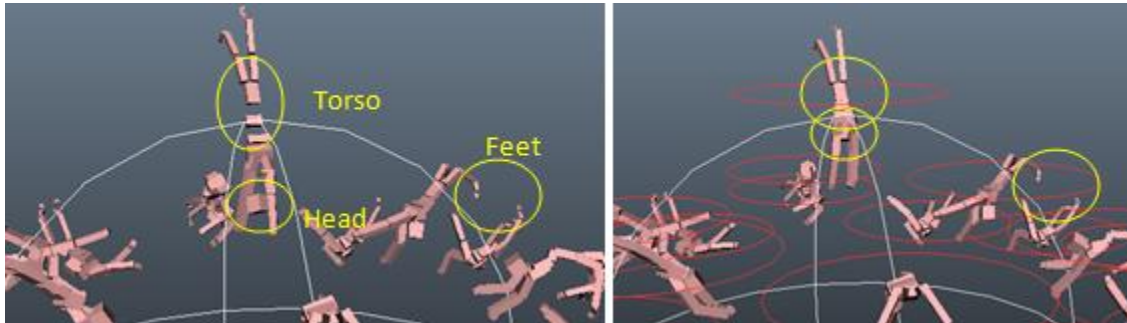


The cache files in d:/abc folder

Agent Cache for Bone Rotate Only

Default, we only record the bone rotate info to the cache files. It can save lot of time in the process of reading and writing.

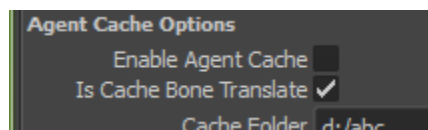
And there is a special usage for cache files. In dynamic simulation, sometimes for some agents, the simulate result is not very precise. Some joints will not convergence very good. But since our cache is not going to record the translate data, after writing out the cache and when the agents read cache file back, the joint separate situation will disappear automatically.



(left) simulation time and separated, (right) read cache from file and convergence

Agent Cache with translate

In some cases (we listed them below) we need record the bone translate info. For storing the translate info on the bone, the only thing we need to do is enable “isCacheBoneTranslate” feature in Miarmy Global before creating cache.



Enable record translate before making cache

The situations we must enable record translate:

- Logic channel contains offset translate
- Some parts separated from agent when dynamics enable (like drop sword and shield)

Agent Cache with cloth

If there are clothes on agents, our system will automatically record them.

Agent Cache Format

Miarmy cache is a binary stream, which contain the format.

```
<numberOfAgent><numberOfCloth>(<numberOfIndex1><numberOfVertex1><numberOfNormal1>)<Index><Vertex><normal>.....<numberOfIndexn><numberOfVertexn><numberOfNormaln><Index><Vertex><normal>)<agent1 root>,<agent 1 bone>,  
.....  
(<repeat cloth n>)<agentn root>,<agentn bone>
```

Out engine will compare the agent number between the first data in cache file and the number of agent in scene, and if they are not equal, the cache will not be read and setback pose to agents.

Mesh Drive

Concept

One can generate geometries for each one of agent directly in Maya, and use Miarmy Agents to drive these geometries. These geometries are driven by agents rather than the regular ways such as skinCluster joints or geometry caches, and all the processes are happened in Agent memory layer instead of Maya Script layer, so it is relatively faster than regular deformation method.



Driven Mesh Rendered by Mental Ray

Features:

- The Driven meshes are actually Maya meshes and can be rendered by any type of renderer or plugins directly in Maya! Such as Mental Ray®, V-Ray®, Renderman for Maya®
- Interactive display geometries, OpenGL or viewport 2.0
- In not far future, these meshes can be applied Blend Shapes feature

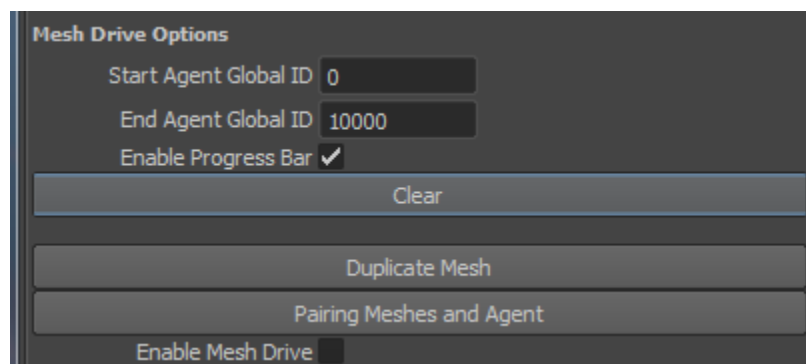
Optimizing Mesh Drive before Implementation

As you see, Mesh Drive indeed creates geometries (Maya Mesh Node) in scene, so it can be very slow if your character is complex or number is too large. It is actually a render feature instead of animation feature so we provide the following suggestions for optimizing your crowd scene:

- Linux is highly recommended
- This feature is very memory consuming, please use the machine with big mount of memory (more than 4G, and recommend 8G+)
- Make Agent Cache before using mesh drive
- Make your mesh simpler if you have large number of agents
- Keep down the geometry number for each single agents (each agents may contain several geometries)
- Change the start and end agent ID for partial display agent geometries
- **Warning: this feature would be relatively slow in Maya 2011 Windows version, but faster in Maya 2012, and the fastest on Linux System.**

Generate (Duplicate) Mesh

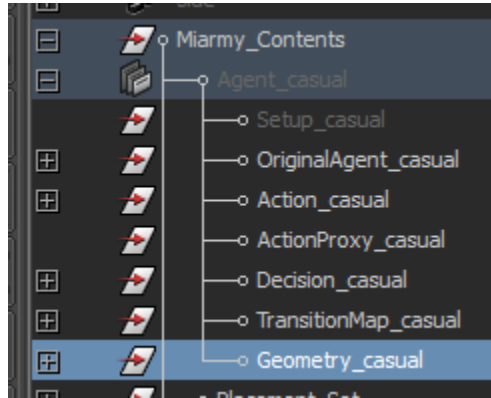
In this process, our script will duplicate all of the geometries for each type of agent, and optimize them, and then duplicate for each placed agent with the random geometry rules. All the process is automatically.



Mesh Drive Feature GUI

Steps breakdown:

In the first step, system will duplicate the Geometry_<AgentName> group for each type of agent and rename them (add MDG_MDG_ prefix), then put them out of Miarmy_Contents. We called them “template groups”. The structure inside of this group is naturally same as the Geometry_<AgentName>



The geometry group in Miarmy Contents to be duplicated



Step 1: Duplicate a copy out and rename optimization automatically

In the second step, system will automatically delete the all of the history and intermediate shape nodes for all of the geometries in the “template groups”.



Step 2: the grey ones, the trash intermediate shape nodes need to be deleted

In the third step, system will use random geometry rule to duplicate geometries for each one of agents, and group them into a new group MDGGrp_<AgentName>. You may notice the geometries are arranged sparsely without any hierarchy because they are the already selected ones by the random geometry rules.



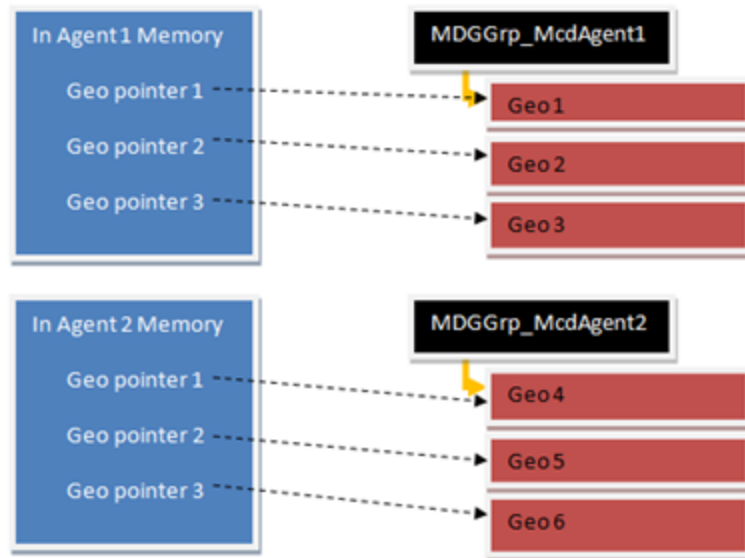
Step 3: duplicate meshes for each of agent obey the random rule we talked able in previously

In the last step, system will pair geometries in agent memory <AgentName> and MDGGrp_<AgentName>, please check out the Pairing session below

Pairing

In this session, system will pair the geometry pointers in agent node to the mesh nodes in scene. After this process, system will not have to re-search the geometries from crowded scene for agents again in time iteration. And the speed will be much faster. Technically, it is a process establishes a map relationship between the memory of agent and the scene meshes.

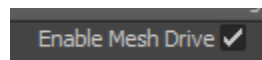
Please take a look at the picture below. After this process, the pointers in memory of agent will point to the real meshes in scene.



Agent -> Maya Geometries Memory Mapping

Enable Mesh Drive

If the mapping relationship has been established, each time the agent pose been updated, the meshes in scene will be deformed by the Miarmy agent.



Left: dynamic + force field, Right: viewport 2.0

Render Solutions

There are 6 types of render solutions for Miarmy, the renderman plugin and mesh drive.

- Renderman pluing: using 3delight API, we dedicated designed a renderman plugin for exporting scene to 3delight renderer or to the RIB file, with procedural primitives.
- Mesh Drive: using this solution, you can use arbitrary renderer to render you scene out. However, this method will actually duplicate render mesh out in Maya scene and its memory consuming.
- Mental Ray:
- V-Ray:
- Alembic Geometry Cache
- FBX Export

Since we already introduced the Mesh Drive in above, from here, we will focus on the renderman plugin solution.

Render by Renderman

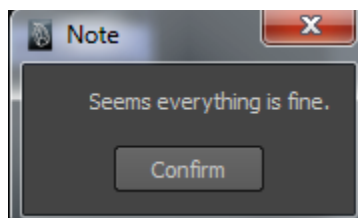
For rendering image, we dedicated provide a render plugin for previewing image, rendering image sequence or exporting RIB file. This plugin is based on 3delight API, so you need install “3Delight Studio Pro” before using these features.

Download from the following link and only 64bit:

http://www.3delight.com/en/index.php?page=3DSP_download

3Delight Configuration

Before using Renderman plugin, you need install 3delight correctly. You can easily check whether your 3delight correct by click Miarmy > 3delight Setup > Check Renderer Status. If the result is not fine, please reinstall 3delight and reinstall Miarmy can solve this



Result is fine

The check render status script will check the following things:

1. Whether The 3Delight.dll in Maya bin is correct version
2. Whether install 3Delighth
3. Whether install correct version of 3Delight

4. Check shader path
5. Check whether shaders has been compiled
6. Check setup DL_DISPLAYS_PATH

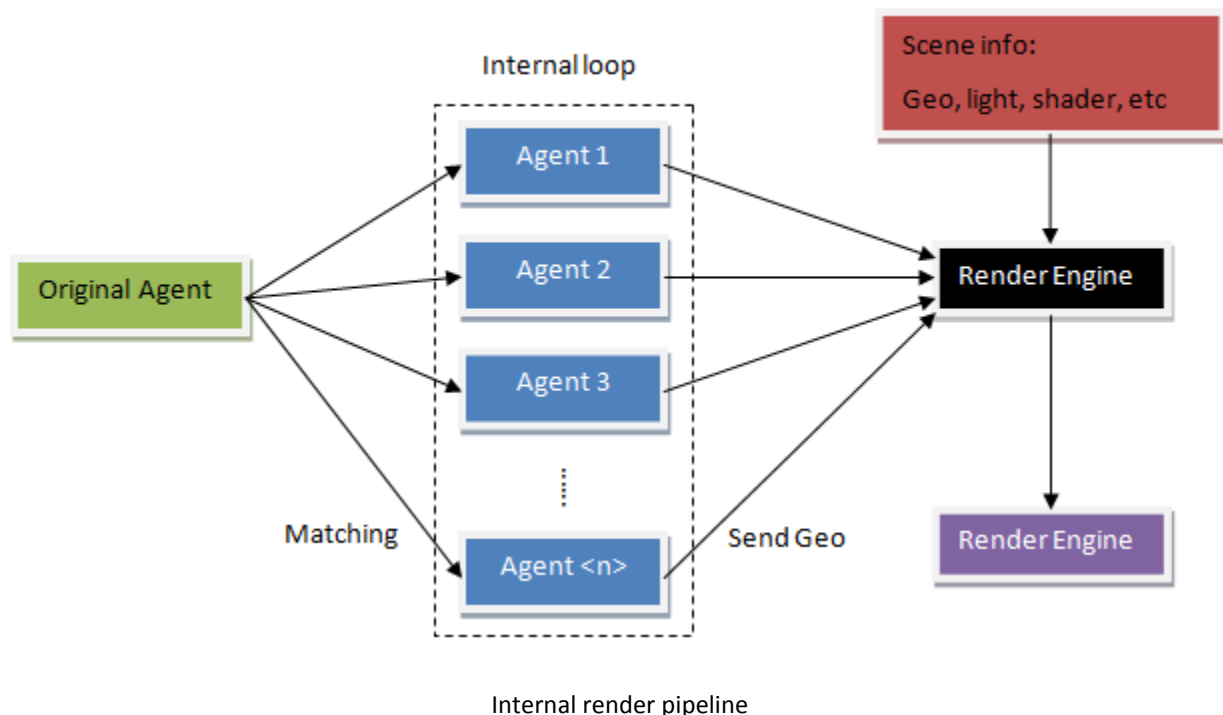
Manually Setup Renderer (Windows)

If you are familiar with the windows system, you can please setup your render by following steps, it should be make your image can be rendered out.

- Close Maya
- Re install 3delight 10 64bit firstly
- Copy the 3Delight.dll from 3delight bin folder (e.g. F:\Program Files\3Delight\bin) to the Maya bin folder (e.g. F:\Program Files\Autodesk\Maya2012\bin)
- Open Maya and load Miarmy
- Click Miarmy > 3delight Setup > Compile Shaders

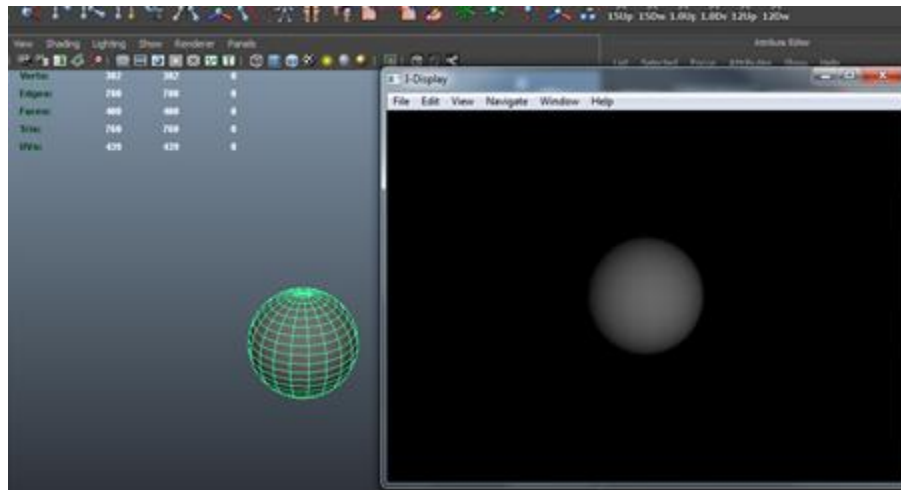
Pipeline

The render pipeline is fairly simple and clear. There is an internal loop based on all the agents in scene, the original agent will match to the agent in current iteration, and then export the geometries from original agents to the engine, finally we will send the scene information like the geometries, lights, shaders and so on to the render engine. The engine will render out the image based on the scene description info.



How to Render

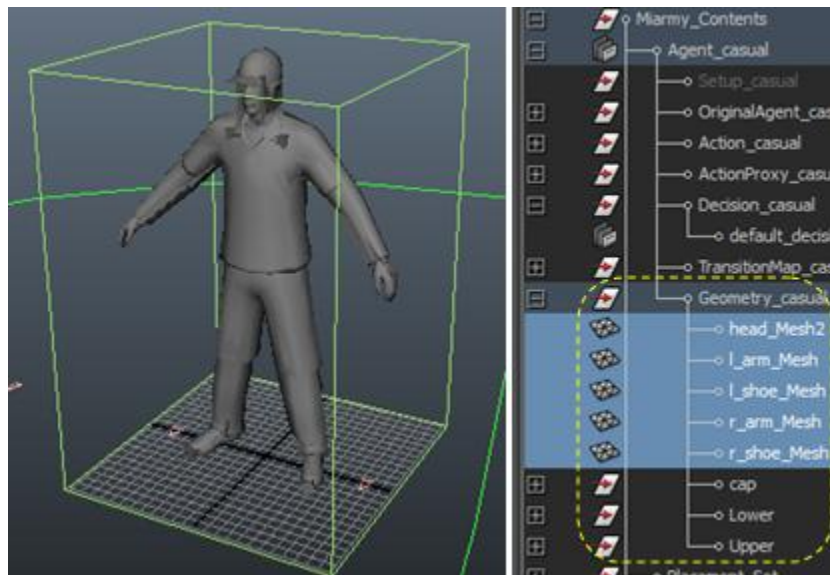
If the renderer setup correctly, you can create something and light them up then click Miarmy > Render Preview (i-display). Once the stuff rendered out, it proof that your renderer is ready for duty.



The simple render test

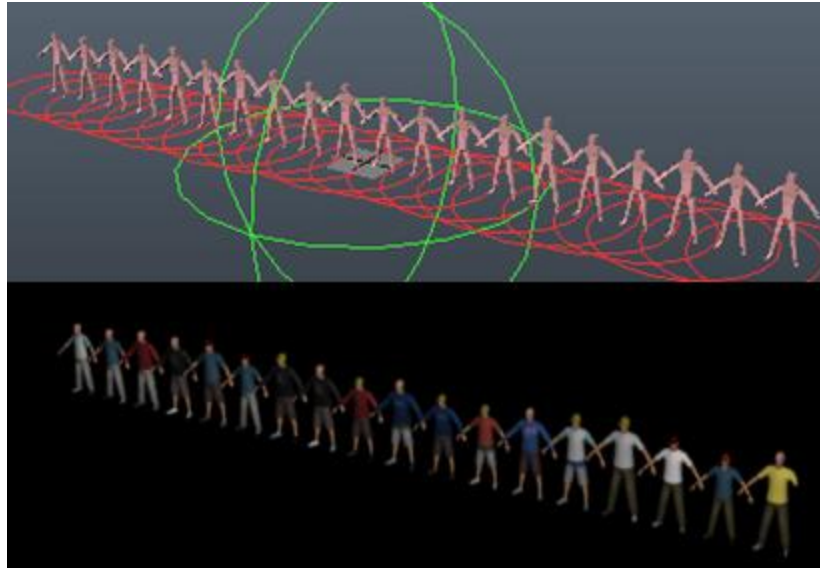
Render Agents

For rendering agents, we need firstly make sure the original agent have geometries been skinned on it. And the Geometry_<Agent Type> group contains these geometries.



Original Agent with geometries skinned on its joints

And you need place the agents out from place node(s).



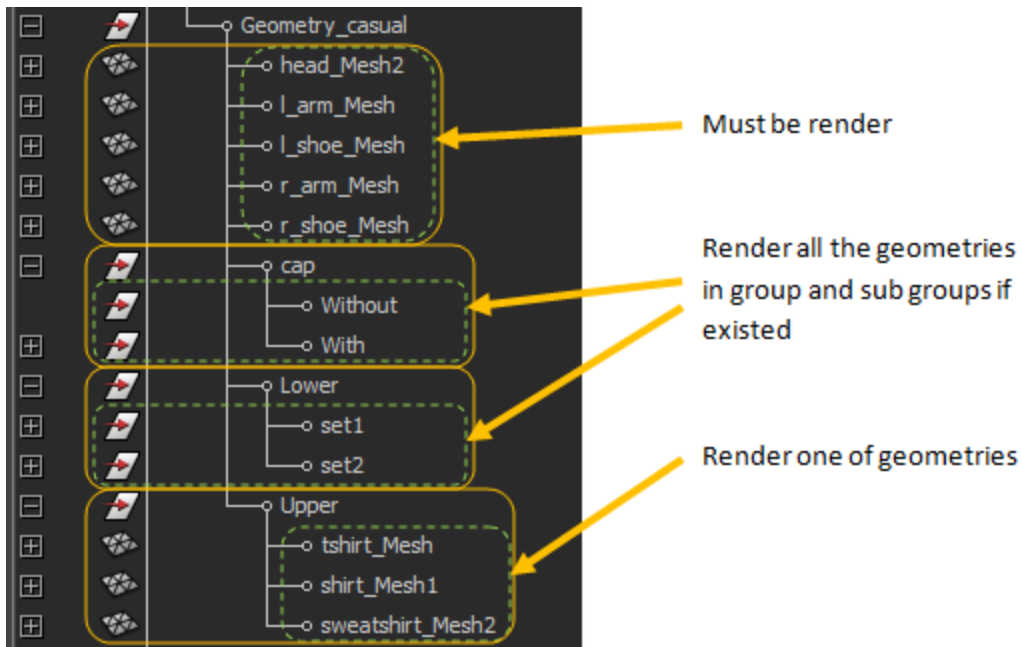
Test render

Random Geometries Rule

We already know that the render engine need fetch the geometries in Original Agents. We can easily re-arrange the structure of these geometries in Geometry_<agent type> group for achieving randomizing.

There are 3 rules for making the geometries random:

1. The geometries directly under the Geometry_<agent type> will be always rendered.
2. And if there are several sub groups, and these sub groups are children of a root group, and this root group is directly under the Geometry_<agent type>. We will choose all the geometries in one of sub groups even there still have hierarchy in sub group. Notice, there may be no geometry in sub group.
3. If the geometries under a child group of Geometry_<agent type>, one of these geometries will be rendered.



Random rules example. One of the items in green bound will be selected.

Please check out the above picture,

1. The items in the first orange bound will obey rule 1

The system will choose one of the geometries from the green bound. For example, “head_Mesh2”, “l_arm_Mesh”, “l_shoe_Mesh”, “r_arm_Mesh” and “r_shoe_Mesh” will be all selected.

2. The items in the second and third orange bound will obey rule 2

The system will choose one of the groups in green bound, and then fetch all the geometries out from that group. If that group doesn't have geometry, the system will not fetch anything, whereas if that group has further sub hierarchy, the system will fetch all the geometries from all sub hierarchy, for example, in “set1” will be chosen, and all the geometries in “set1” and sub hierarchy will be all selected.

3. The items in the fourth orange bound will obey rule 3

The system will choose one of the geometries in green bound. For example, the “shirt_Mesh1” will be selected.



Notice the cap, pants and t-shirt geometries

Texture Naming Convention and Random Textures

We can randomize the textures just by the texture name. If you make your texture names by the following rules, system will random texture for your geometry automatically without any further setup.

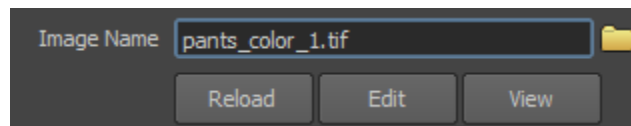
Correct naming convention for automatic random texture:

<Texture Name>_<padding>.<format>

1. Need at least an underscore before numbering
2. Only single padding number
3. Sequential numbering

If your texture names are correct, you just need fill the **first** texture name in the Image Name attribute.

Example: if you assign “**pants_color_1.tif**” in your Image Name attribute, the system can automatically parse out the **Green name** in the following example.



Assign image name in texture file node

- pants_color_1.tif
 - pants_color_2.tif
 - pants_color_3.tif
 - pants_color_4.tif
-
- pants_color_1.tif
 - pants_color_2.tif
 - pants_color_0.tif (cannot read, not sequential)
 - pants_color_4.tif (cannot read, not sequential)
-
- pants_color_1.tif
 - pants_color_2.tif
 - pants_color_03.tif (cannot read, not right numbering)
 - pants_color_04.tif (cannot read, not right numbering)

Incorrect naming:

- `pant_color1.tif` (no underscore, cannot reach)
- `pant_color0001.tif` (no underscore, cannot reach)
- `pant_color_001.tif` (non-single padding, cannot reach)

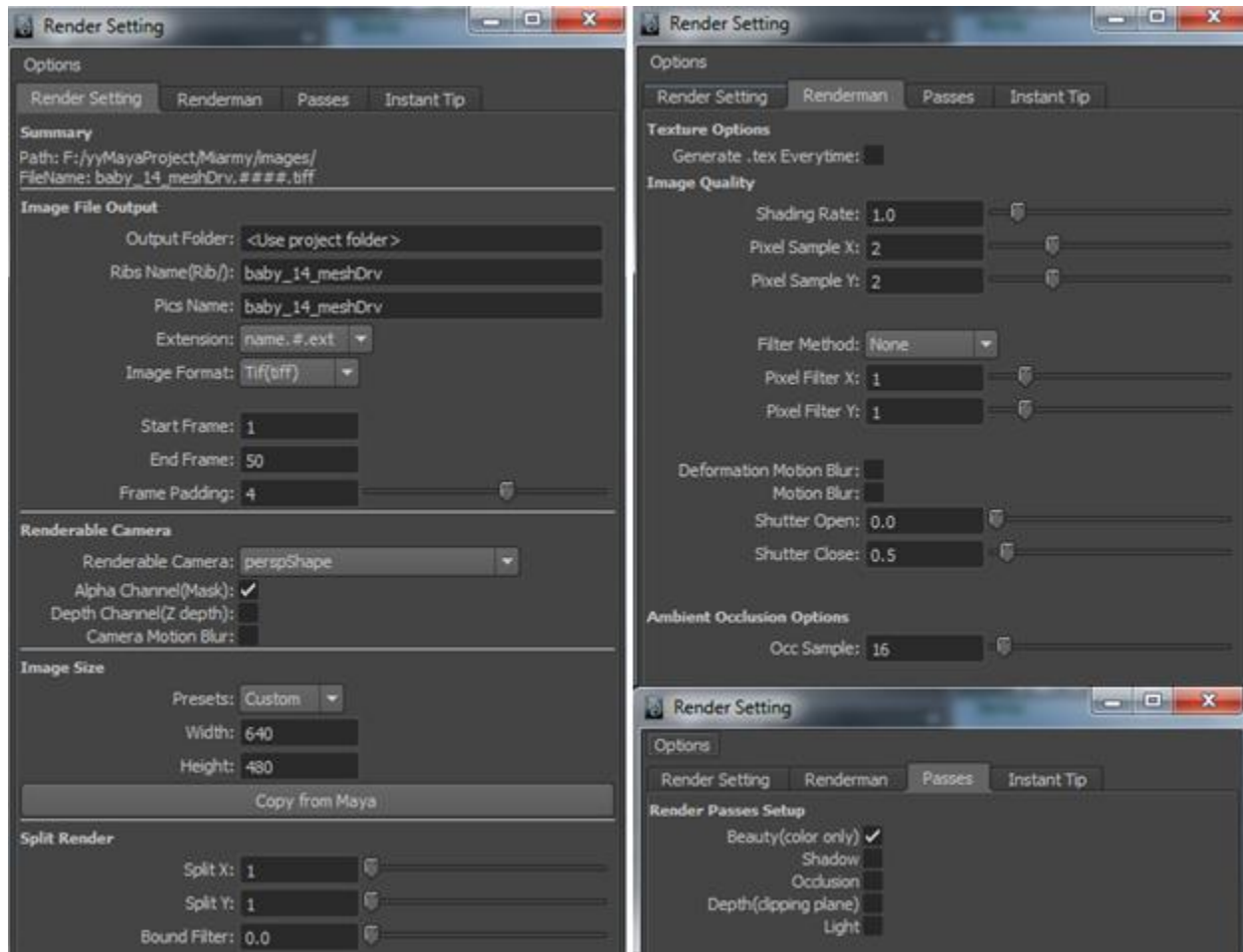
Once we parse out all the possible textures, for example, from “`pants_color_1.tif`”, we found “`pants_color_2.tif`”, “`pants_color_3.tif`” and “`pants_color_4.tif`”. Then, the system can automatically random select one of it based on the agent. For example, “`pants_color_2.tif`” will be selected. And the “`pants_color_2.tif`” will be sent to the render engine.



Random textures

Render Global

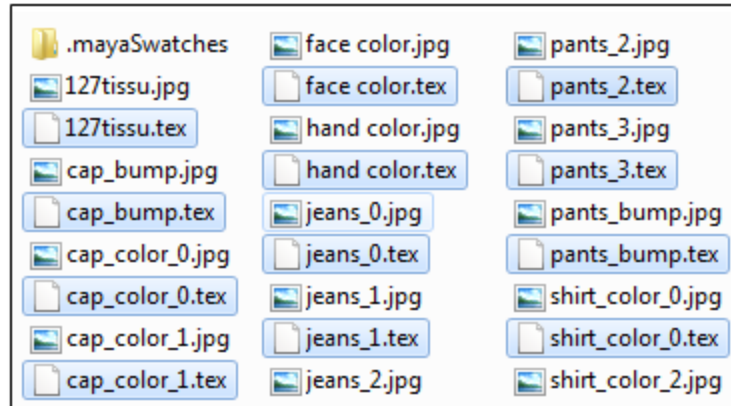
Once you know how to render the image out, you can adjust the attributes in Render Global for fine-tune the image quality.



Render global

Texture generate and path accessible

Due to that the Renderman need generate the .tex file for each of texture, please make sure the directory you put your textures is writeable. Or you need pre-compile the ".tex" file and put these file on sever. Please notice the following figure. You may notice each texture file has a ".tex" file counterpart.

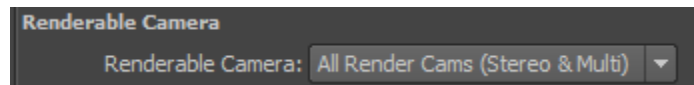


The .tex file in the same folder of texture

Multi cam render

Our system will automatically render image from all renderable cameras, if you want to render stereo multi can, please just setup in Maya Render settings.

If you have several renderers cams, the Renderable Camera in Miarmy Render Global will change to “All Render Cams (Stereo & Multi) automatically. And you don’t need to setup them again here.

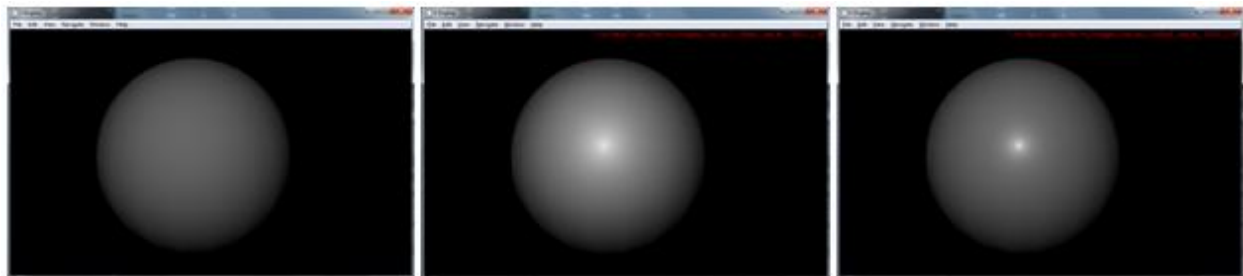


Render multi cam

Preset shaders

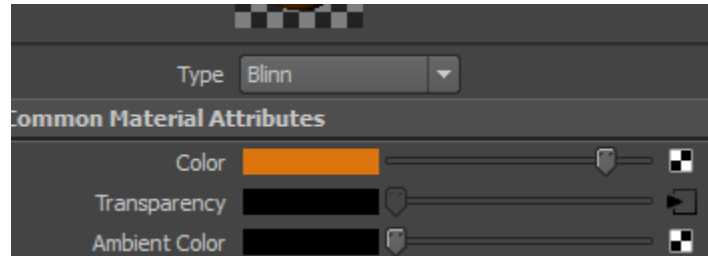
You can directly use Maya shader and its instinct features to render the scene, our render plugin will automatically translate the Maya shader to Renderman shader parameters, then render image out. Just like 3Delight for Maya or Pixar’s Renderman for Maya.

We only support 3 types of preset shader of Maya. They are Lamber, Blinn and Phong. You can use Miarmy > Rendr Preview (i-display) for directly rendering image out.



Lamber / Blinn / Phong rendered by 3delight preset shader

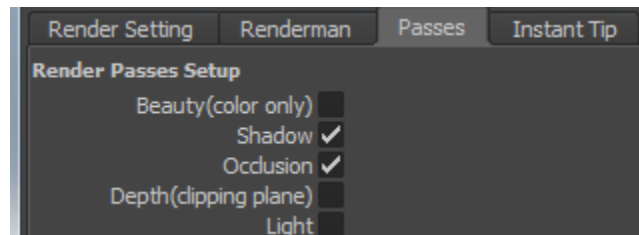
Preset shader supports the following feature, and all of them don't need you setup, just give them Maya Shader:



Only need setup Maya shader

Note: If you render image by Miarmy > Render Preview (i-display), the preview image will be the first pass in Miarmy Render Global. Whereas if you render images by batch Miarmy > Render Extra > Render batch to Image Files, the render plugin will render all the images in each pass.

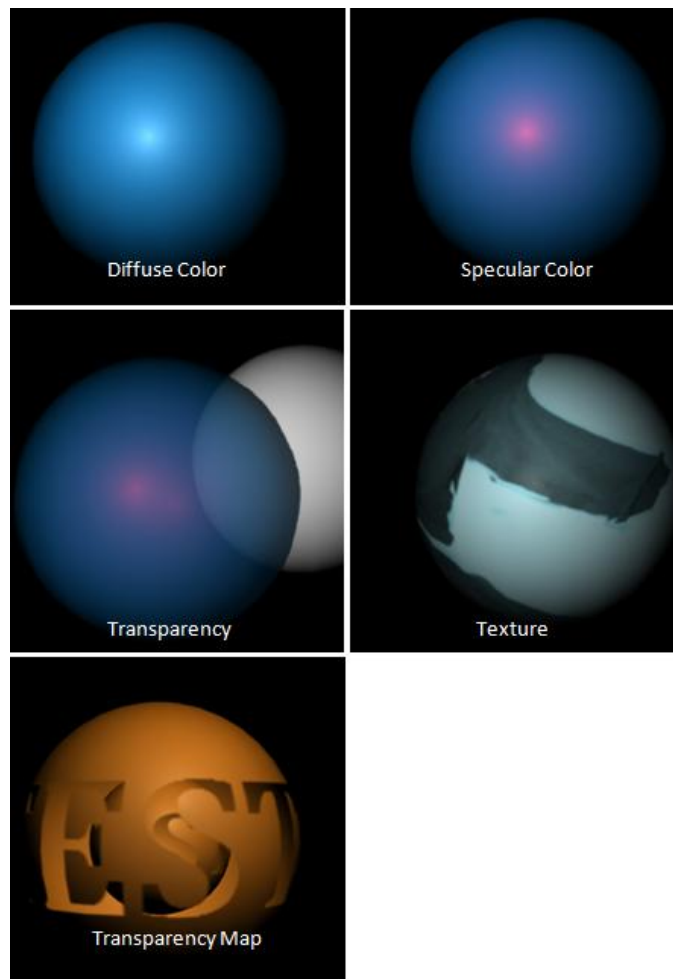
For example like the following picture, if you check Shadow and Occlusion passes, and when you preview render, only shadow pass will be rendered out for current frame. And if you render image batch to files, all the images for every frames and every passes will be rendered to specified path.



Render passes in Miarmy Render Global

Beauty Pass

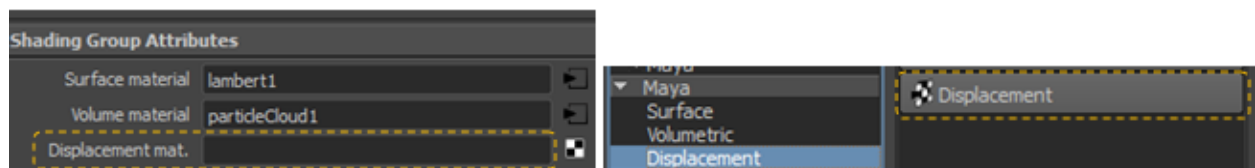
Our system directly support these shader features:



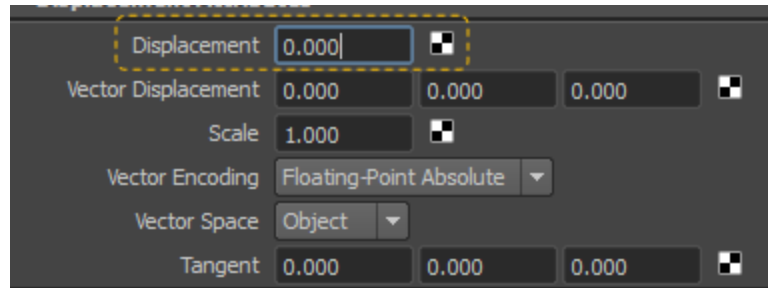
The preset shader feature

Displacement Map

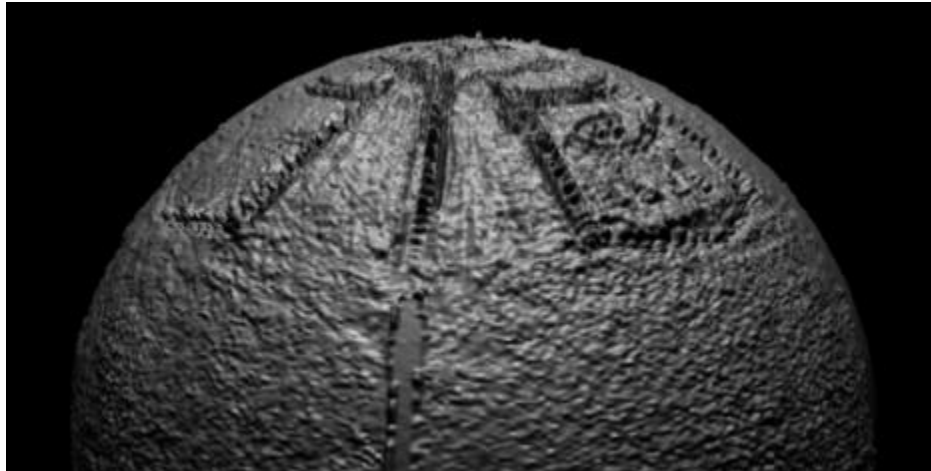
Miarmy render plugin doesn't support bump map. Instead, it supports displacement map. The only thing need to do is link a Maya displacement material to the shading group node. Because Renderman based on micro polygon, the displacement map is faster and better then bump.



Creating displacement material



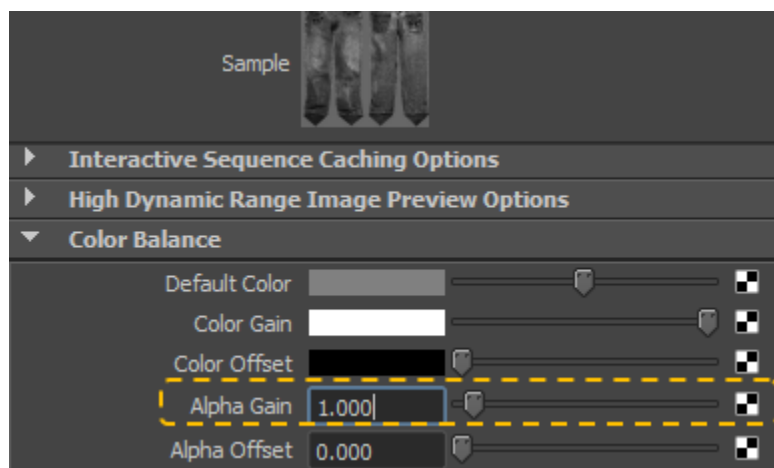
Fill the displacement map

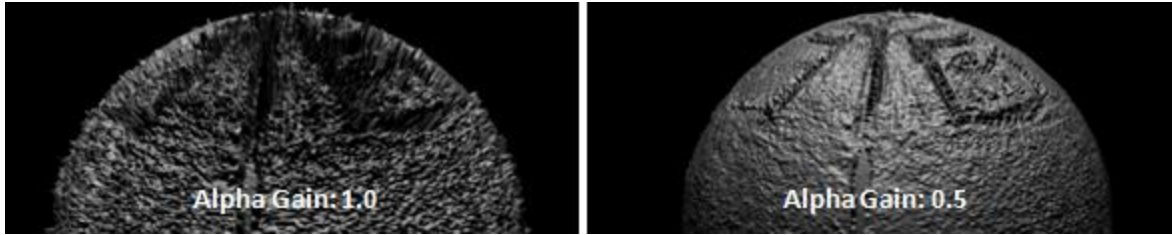


Displacement map

Displacement Map Intensity

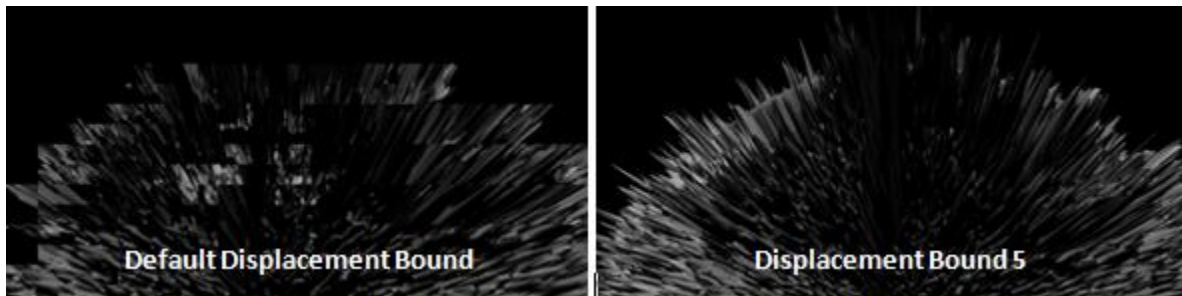
You can directly change the alpha gain parameters for tweaking displacement intensity:





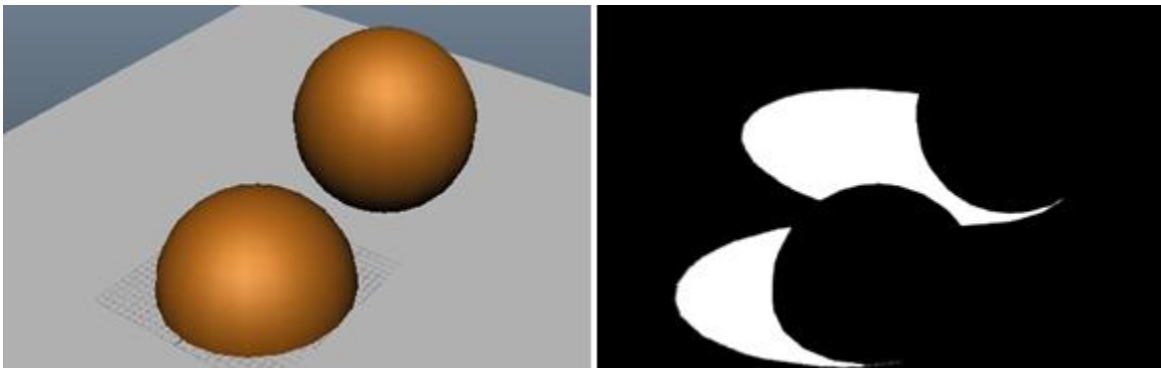
Displacement Bound

If you are using high intensive displacement, there may be some render error on it like the following left picture. For solving this, you need add a displacement bound attribute to the geometry. Miarmy > Render Extra > Add Displacement Bound Attribute



Shadow Pass

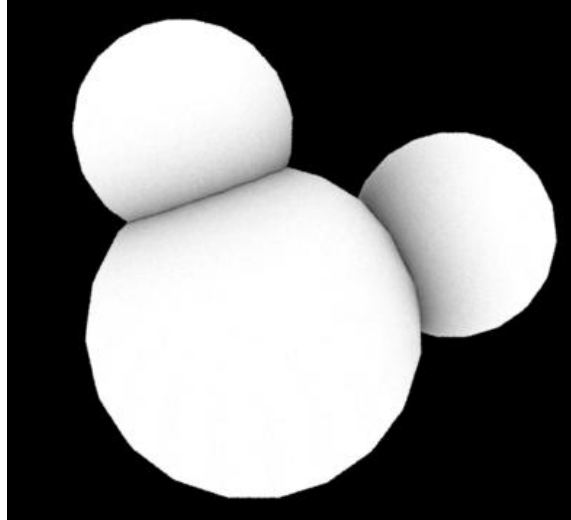
You need enable depth shadow map in light attributes



(left) maya viewport (right) Shadow pass

Render Ambient Occlusion:

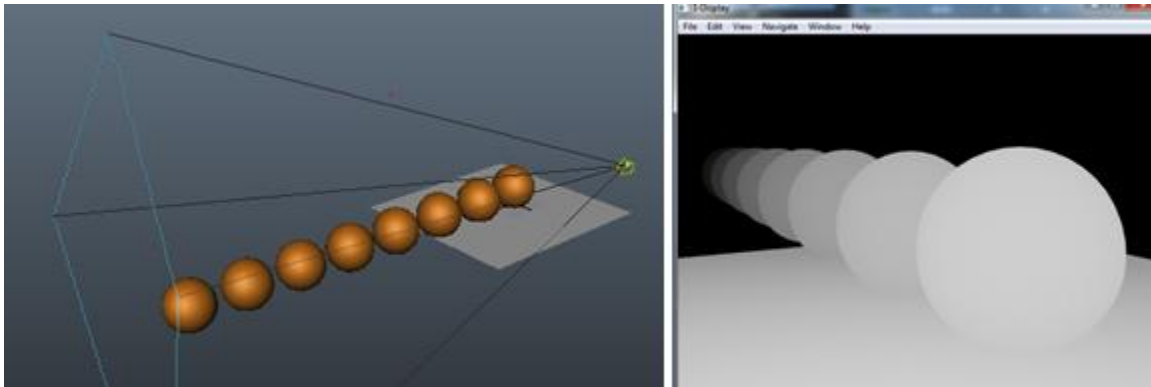
The occlusion quality can be setup in Render Global OCC samples



Occlusion pass

Depth Pass

Controlled by near and far clips



Depth shader controlled by camera near and far clip. (Left) frustum (right) render result

The shader located in 3delight shader path. They are:

- McdBumpy.sdl
- McdDOF.sdl
- McdGatherAO.sdl
- McdSpotlightInv.sdl
- McdTxtPlastic.sdl

And the source codes are located in Miarmy installation place.

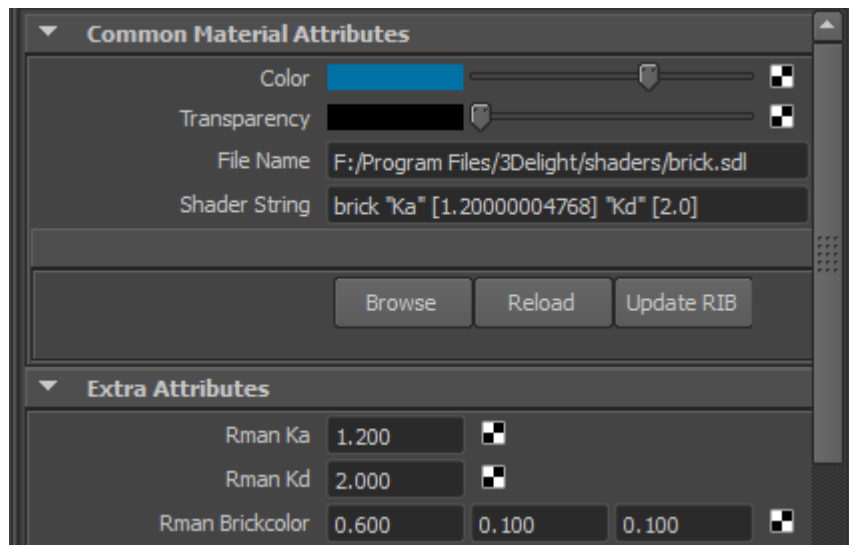
Self-defined Renderman Shader

You can create Renderman shader shell and link them to external renderman shader. In current version, we only support

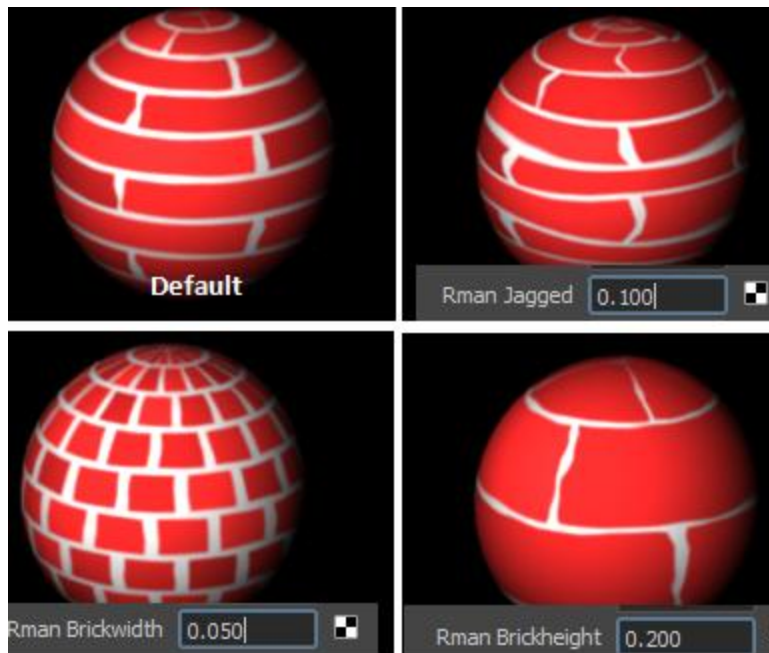


RM shader shell

- Click "Browse" to select shader,
- Expand "Extra Attributes" for displaying the attribute of the shader
- Click "Update RIB" for updating the "Shader String".
- Click "Reload" for clear all attributes and set to default.

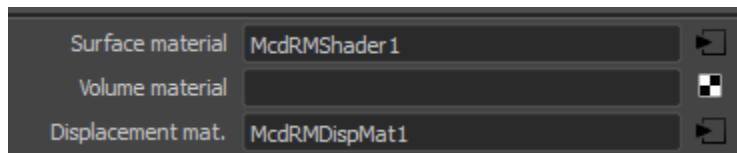


The emboss shader example

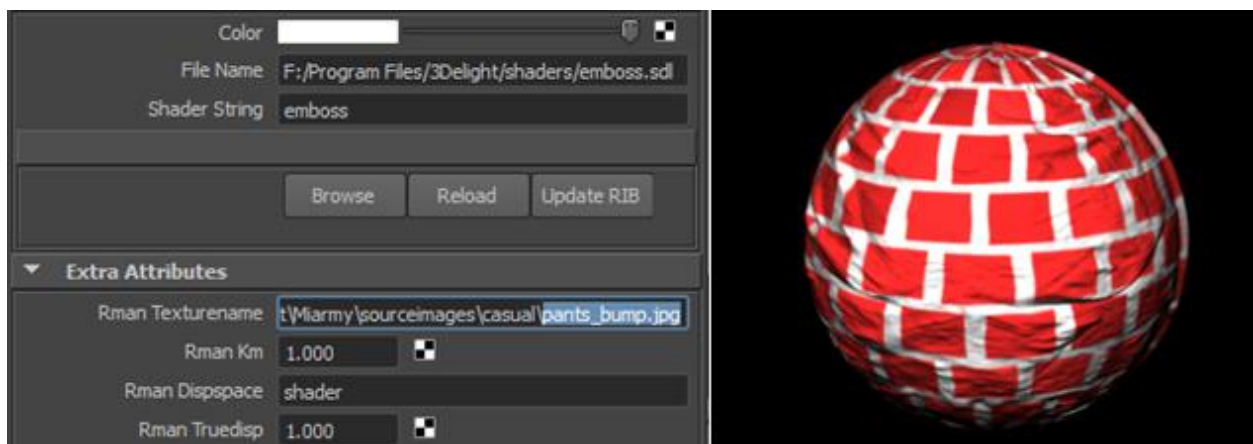


The render result from “brick” shader with different parameters

Displacement Map is the same as surface map, but it need connect to “Displacement mat.” attribute.



Connect a McdRMDispMat shader to node



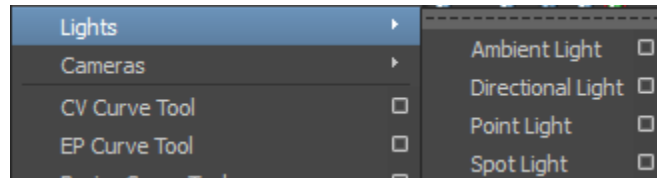
Displacement map and render result

Light Usage & Shadow

Lights

Miarmy support 4 types of lights for lighting.

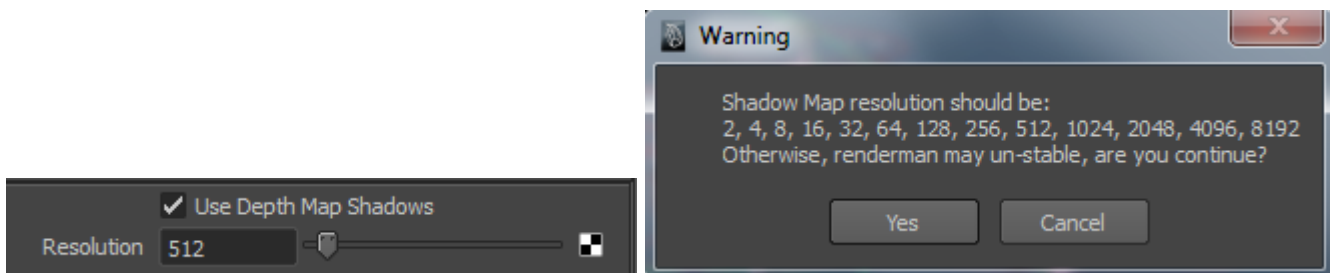
But only support Spot light for shadowing.



Available lights

Shadow

Need enable shadow Map and resolution should be 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, and 8192.



Need enable shadow Map and resolution should be some specific value

Motion Blur

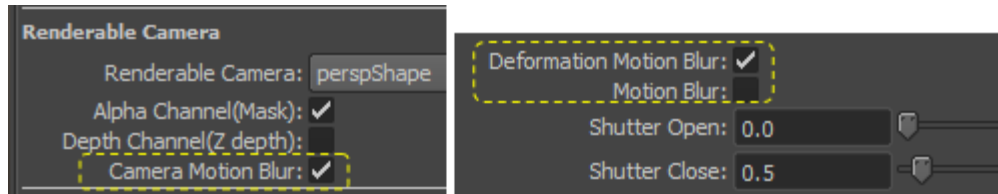
Notice: motion blur only can work with agent has cache

There are 3 types of motion blur in Miarmy renderman plugin

- Camera motion blur: blur when camera moving
- Transform motion blur: blur entire agent by its root movement
- Deformation motion blur: blur based on the deformation of the geometries

Deformation motion blur have more accurate and better result whereas transform motion blur is faster, usually you need choose one of them but don't enable them both.

The first is a camera attribute and you can enable it in render setting page:



3 types of motion blur in Miarmy Render Global



Camera motion blur (camera is rotating)

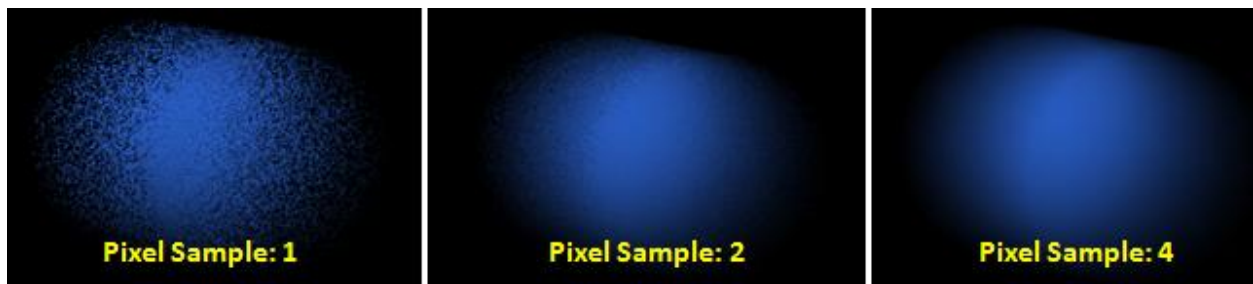


Transform motion blur (blur entire agent)



Deformation motion blur (blur only deforming parts)

You can change the motion blur quality by Pixel Sample attributes in render global



Split render

Reason

When the scene contains several thousand agents and each agent contains several geometries, our machine may meet the bottleneck. At this time, we need separate the render job to several parts and finish each part one by one and finally combine the results. We do this using split render.

Under the hood

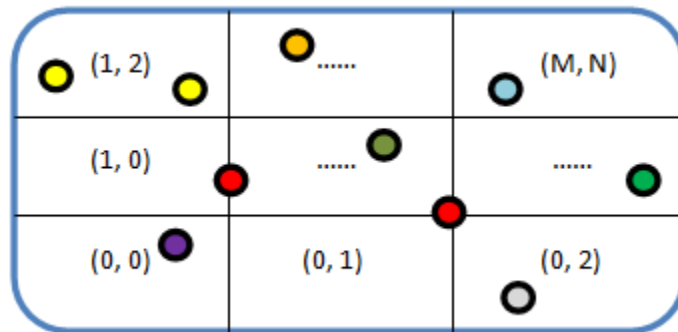
Split render is actually going to split screen to different parts and render each parts one by one, and each part will output a picture. Then, we need combine them in post-production software. The splitX and splitY attributes are used to specify how you need split the render screen.

- The calculation is based on bounding box of the agent, in original agent.
- The agent with **break dynamic bones** (such like the solider dropped the sword) will be rendered in every split part

For example in following 2 pictures, we split the screen 3 by 3 to 9 parts, and our renderer will render each part of them.



Split the render screen 3 by 3 to 9 parts



Split screen and separate the agents

Render Result

The render result naming convention will be:

<Frame Name>_<pass>__<part X>_<part Y>_<Frame Number>

The image can be rendered out fast also without any bottle neck:



The result of split render

Procedural Primitives

Reason

The reason of using procedural primitives is pretty simple: save space of hard disk.

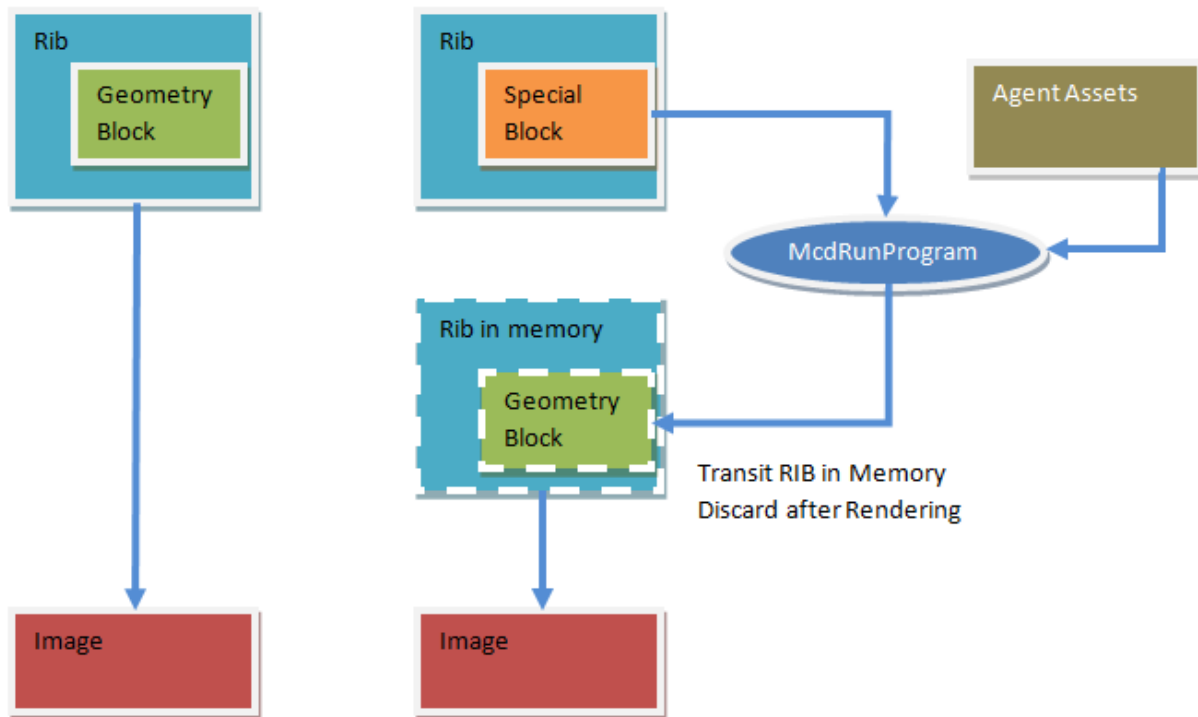
Under the hood

Traditional Renderman RIB file will contain the all geometry details in file for rendering the scene, each single geometry will be included in each frame of RIB, sometime this will using huge amount of hard disk, for example you have 10000 agents with 100000 piece of geometry it's impossible store them in hard disk directly.

Procedural primitive is a feature can insert these geometries information in render time and discard them automatically after rendering. So it will save your hard disk, but not accelerate your render.

The Pipeline

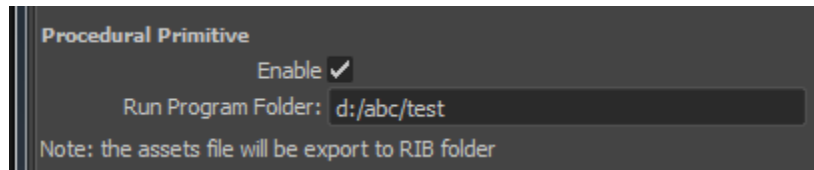
1. We firstly export some information on hard drive, we call them agent assets. The agent assets actually are some information of "Pre Bind" pose and geometries.
2. Then export RIB file with special block instead of geometry block (means, the RIB will not contain geometry information any more)
3. When rendering, system will call a McdRunProgram to read that special block in RIB and generate right geometry info and replace it with the geometry block.



Left: transitional rib approach, Right: procedural primitive approach

Agent Assets Content

The agent assets will be automatically export to hard disk when export RIB and its location will be the same as the RIB



They are 3 types of files

1. Agent Main File:

- a. Agent bone number
- b. Bone names and structure
- c. World inverse matrices of each bone

2. Original Geometry Files:

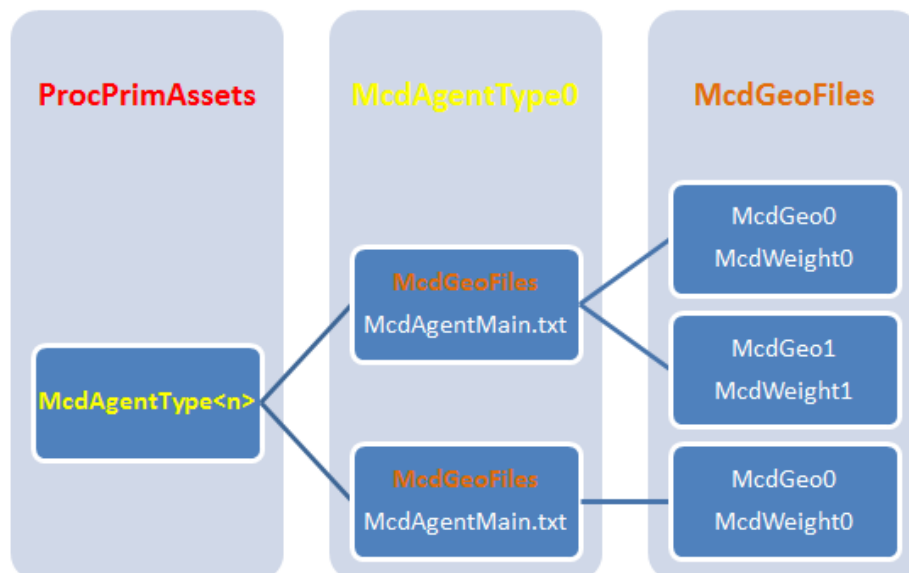
- a. The rib file of original geometry

3. Weight List files:

- a. Each line contains current point weight list (bone id and bone weight)
- b. (the number of lines equals to point number)

Agent Assets Structure

Each scene contains several agent types, each type have severl geometries.



The colorful words mean folder and white names mean files

Rib Tags

Format:

Procedural "RunProgram" ["<RunProgram Folder>/McdRunProgram" "MiarmySession: <agent assets folder>/ProcPrimAssets <agent type id> <geometry type id> <motion blur> <shutter open> <shutter close> <pose data[12] * number of bone>] [<xmin>, <xmax>, <ymin>, <ymax>, <zmin>, <zmax>(bounding box info)]

Example (with motion blur and 2 pose):

```
Procedural "RunProgram" [ "D:/Users/Yeah YANG/Documents/Visual Studio
2008/Projects/McdRunProgram/x64/Release/McdRunProgram" "MiarmySession: d:/pp/abc/ProcPrimAssets/ 0 0 1
0.000 0.500 0.83873 0.00000 -0.54454 0.00000 1.00000 0.00000 0.54454 0.00000 0.83873 0.34956 0.00000 4.77319 -
0.54429 0.00000 -0.83889 0.00000 1.00000 0.00000 0.83889 0.00000 -0.54429 1.18829 0.00000 4.22864 0.83873 0.00000 -
0.54454 0.00000 1.00000 0.00000 0.54454 0.00000 0.83873 0.69913 0.00000 4.54638 -0.54429 0.00000 -0.83889 0.00000
1.00000 0.00000 0.83889 0.00000 -0.54429 1.53786 0.00000 4.00183 " ] [ -1.0337217 2.0822851 -2.220446e-16
2.220446e-16 3.163023 6.1564216 ]
```

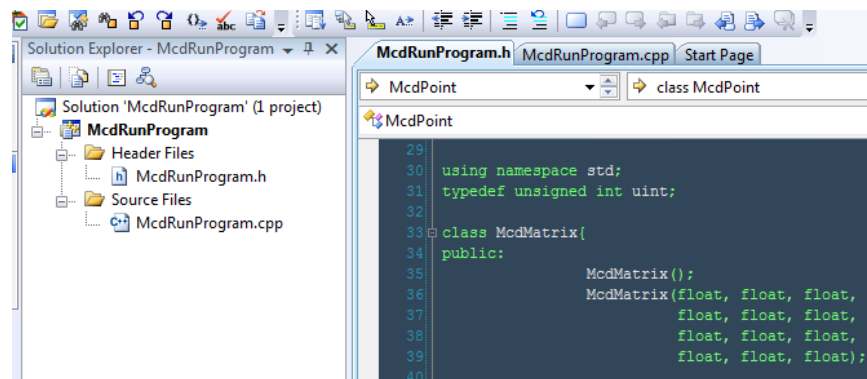
The “McdRunProgram” under the Hood

The “McdRunProgram” is a Renderman plugin can read the agent assets and rig tags, then generate the geometry, and put the result in memory in render runtime.

This program actually implements a very simple skinning algorithm to bind the agent geometry and calculate the right position of each point and right direction of each normal (Note: if using sub-division mesh, no normal need to be calculated). Then put the results to stream (in memory)

Source Code

The “McdRunProgram” is an open source program written by Basefount Team and exclusively open for our formal testers and customers. If you need, please write contact us.



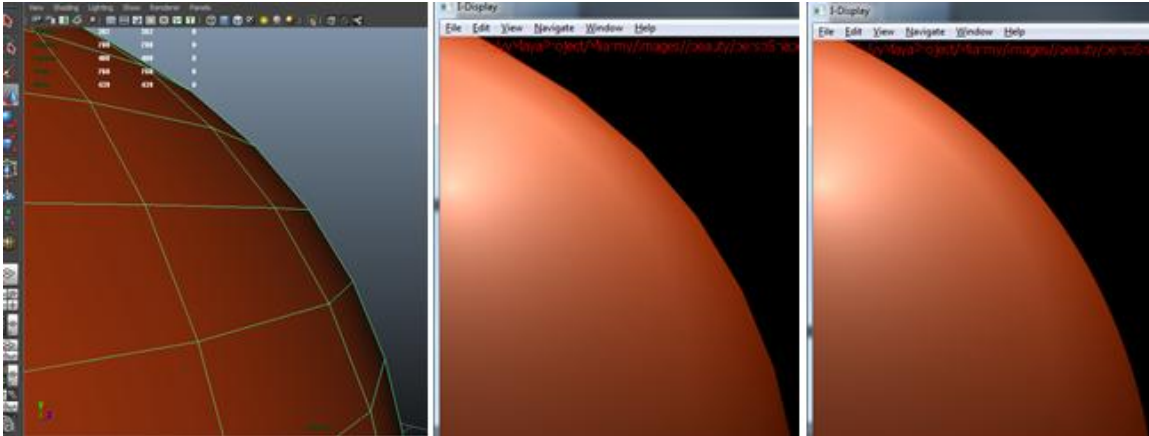
Note:

- The preview render is using procedural primitive automatically and will not generate agent assets
- If you render image from Maya directly, it will automatically use procedural primitive and will not generate agent assets files.

Subdivision Mesh

Subdivision mesh is an powerful feature of Renderman, our render plugin support it!

Once you mark a subd flag on geometry, our renderer will take this geometry subdivision mesh instead of polygon mesh. For adding subd flag to geometry, you need simply click Miarmy > Render Extra > Add Subd Attributes. You may notice the hard edge on mesh will disappear and the antialiasing is perfect.



(left) Maya viewport (mid) polygon mesh (right) subd mesh

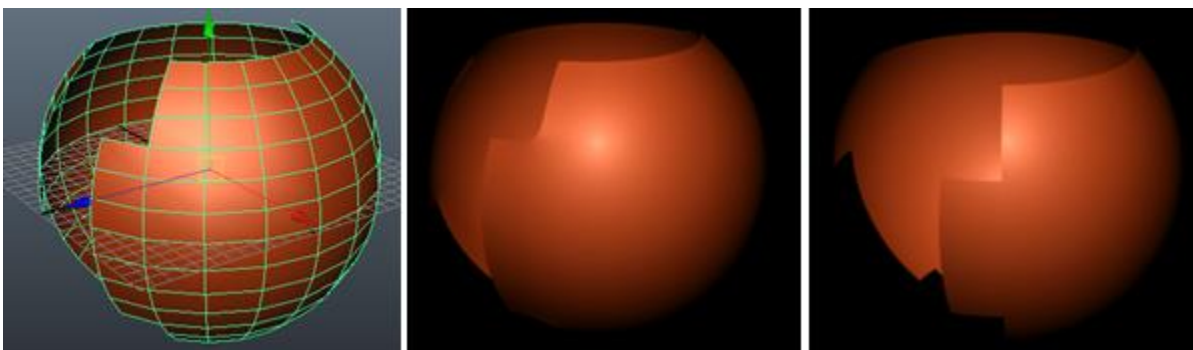
Extra attributes

- **As Subd On**

The object will be rendered as subdivision mesh instead of polygon mesh

- **Interpolate Boundary On**

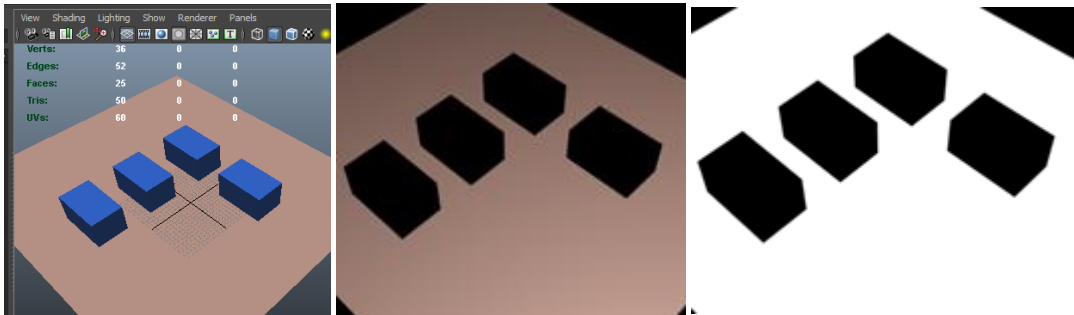
If enable, the subdivision mesh should interpolate all boundary faces to their edges



(left) Maya viewport (mid) disable interpolate boundary (right) enable interpolate boundary

Matte

You can add background color matte to some objects for skip there render but still leave the alpha channel there. You can simply click Miarmy > Render Extra > Add Matte Attribute.

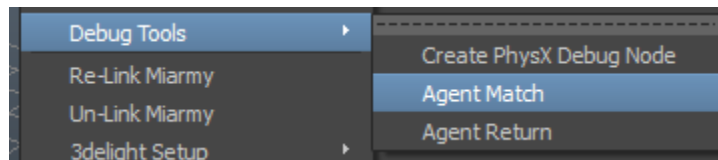


(left) maya viewport (mid) render result (right) alpha channel

Fetch Agent Render Information from Miarmy to Your Own Pipeline

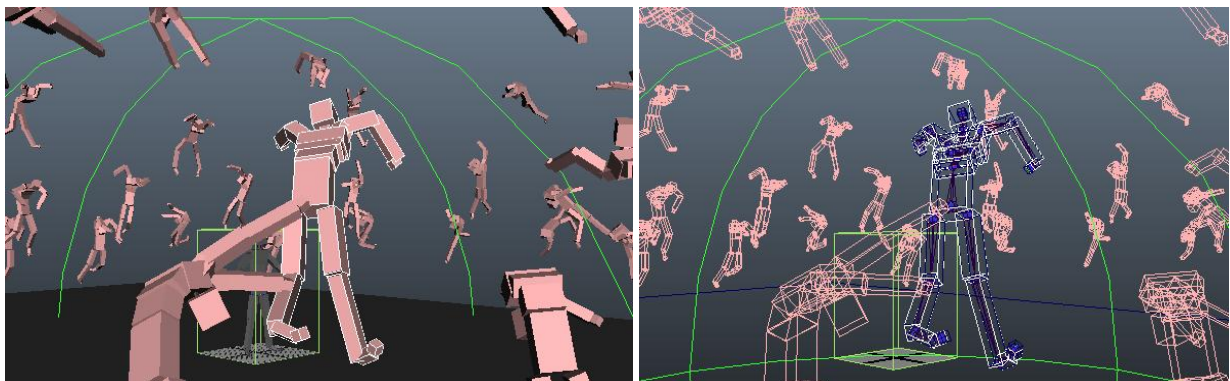
Fetch Joint Information

By menu Item



Select any agent, and click Debug Tools > Agent Match, you original agent will align/match to the agent, no matter cloth or broken joint part, animation or dynamical, the result should be perfect matched.

Click Debug Tools > Agent Return for put your Original Agent to initial pose.



Match "Original Agent" to "agent", then you can fetch joints info from Original Agent

By MEL/Python Command

The agent match process can be done by script,

1. Select any one of agents
2. Run **"McdAgentMatchCmd -mm 1;"** MEL script for matching Original Agent to selected agent

Put the Original Agent back to origin:

1. Run **"McdAgentMatchCmd -mm 0;"** MEL script for putting all Original Agent to origin and T-Pose

Fetch geometry information

By MEL/Python command

Run **"McdGetRenderGeoCmd -rec 0;"** MEL command.

The system will return an array which contents are:

<agent name1><geo1><geo2>...<geoN><'#><agent name2><geo1><geo1><geo2>...<geoN><'#>...

For example:

```
# Result: [u'McdAgent1', u'head_Mesh2Shape', u'l_arm_MeshShape', u'l_shoe_MeshShape',  
u'r_arm_MeshShape', u'r_shoe_MeshShape', u'l_leg_MeshShape', u'r_leg_MeshShape', u'shorts_Mesh5Shape',  
u'shirt_Mesh1Shape', u'#', u'McdAgent2', u'head_Mesh2Shape', u'l_arm_MeshShape', u'l_shoe_MeshShape',  
u'r_arm_MeshShape', u'r_shoe_MeshShape', u'capShape', u'l_leg_MeshShape', u'r_leg_MeshShape',  
u'shorts_Mesh5Shape', u'tshirt_MeshShape', u'#', u'McdAgent3', u'head_Mesh2Shape', u'l_arm_MeshShape',  
u'l_shoe_MeshShape', u'r_arm_MeshShape', u'r_shoe_MeshShape', u'pantsShape', u'tshirt_MeshShape', u'#',  
u'McdAgent4', u'head_Mesh2Shape', u'l_arm_MeshShape', u'l_shoe_MeshShape', u'r_arm_MeshShape',  
u'r_shoe_MeshShape', u'capShape', u'pantsShape', u'shirt_Mesh1Shape', u'#', u'McdAgent5',  
u'head_Mesh2Shape']
```


Part 9 Optimization

This part is a guide for optimizing and speeding up Miarmy in production.

Crowd simulation sometime is a heavy job, brings much uncomfortable frustration. So, the optimization is very crucial for ease your job. In this guide, we are going to explain which part is low efficiency and introduce how to avoid them or take alternative approaches. Also, we provide many useful methods may be faster or better to achieve your same result. We hope this guild can help you to save a lot of your precious time.

Low Efficiency Channels Optimization

Vision

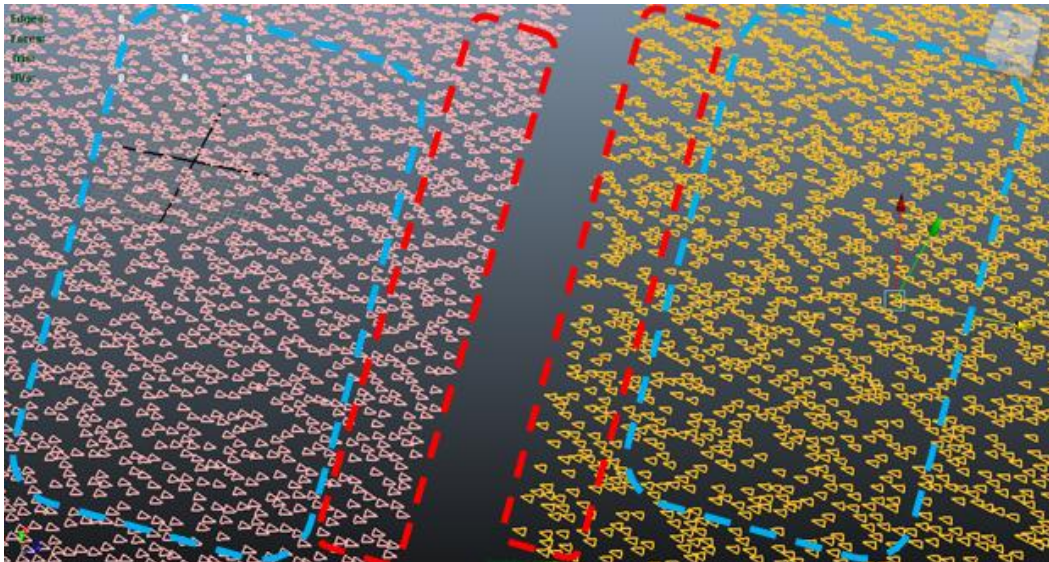
The vision channels have to parse out which agents in its range firstly and this process is exponential type of calculation, therefore, in current version of Miarmy, vision channels are not very high efficiency.

There have some situation you may need vision:

- 2 armies facing together, seek enemies and fight when near
- Follow a target agent or object. Avoid an agent or object.

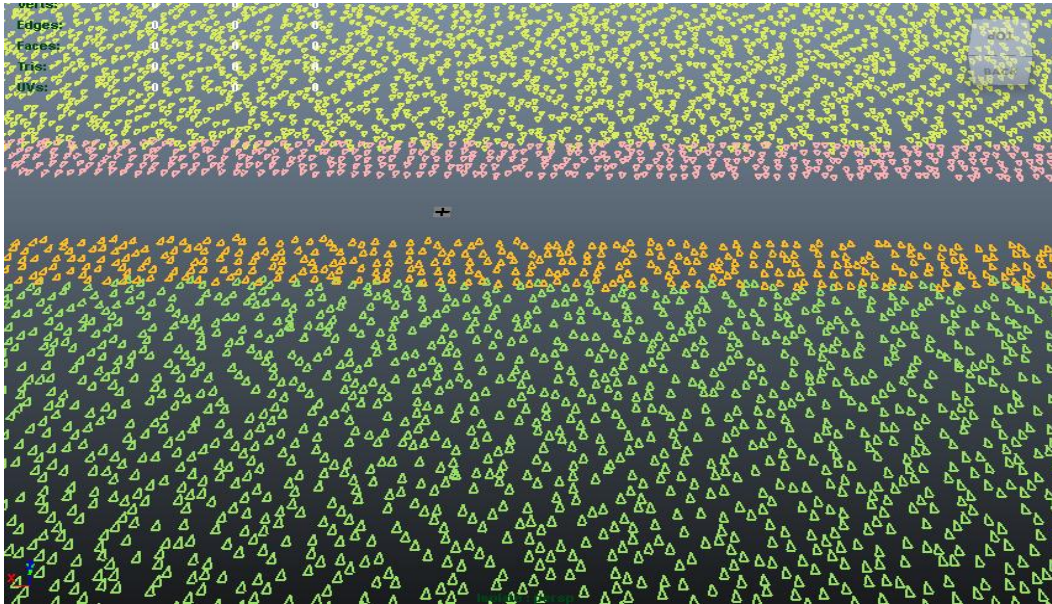
In the first situation, when 2 armies are involved in a battle, there are 2 phases. They are “Rush” and “Fight”

In the rush phase, 2 armies will rush each other, but only the front lines need higher AI for seeking enemies and fight or collide, the rest of agents just need avoid each other. Checking out the following picture, the agents in red bound need higher AI for seeking enemies and fight, whereas the agents in blue bound even don't have opportunity to fight. So we can use DDIM (Dynamic Decrease Intelligence Mechanism) to block the vision for the agents in blue bound. It can tremendously accelerate your simulation speed.



Rush time, disable vision in blue bound

Or we can choose another solution. As the following picture shows, the pink and orange agents in frontlines will have precise collide and fight logic, and the green and yellow agents just need using sound channels avoid each other. This solution is better because you we can even simply the actions and Transition Map for lower AI agents. And simply transition map can save your time.



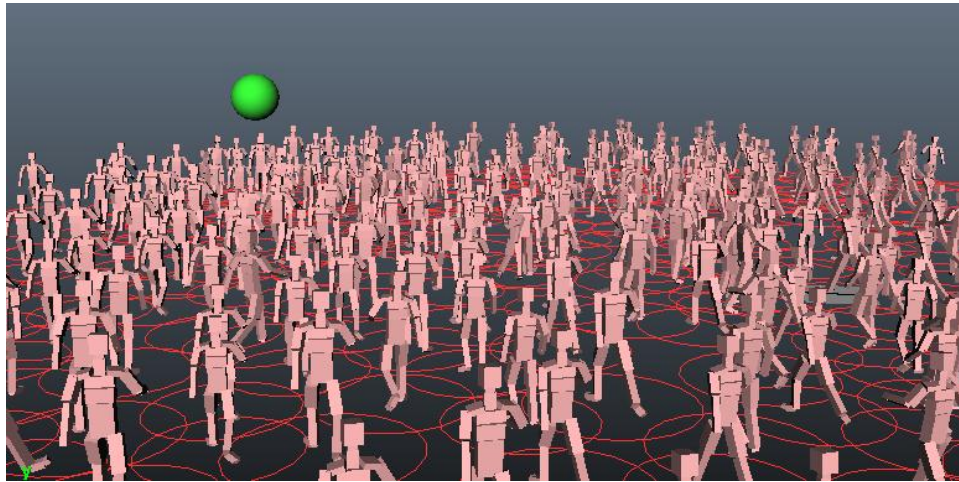
Rush time, pink and orange is higher AI agents whereas green and yellow agents has lower AI

In “Fight” Phase, the agents are near each other we can just use sound channels for detecting and aiming to enemy and controlling fight. Here you can bypass the vision channel directly.



Battle in progressing, can totally disable vision

In the second situation, when the agents are following or avoiding something, we highly recommend make agents to follow/avoid target using spot or zone. Following/avoiding spot or zone will be much faster than the solutions with vision channel.

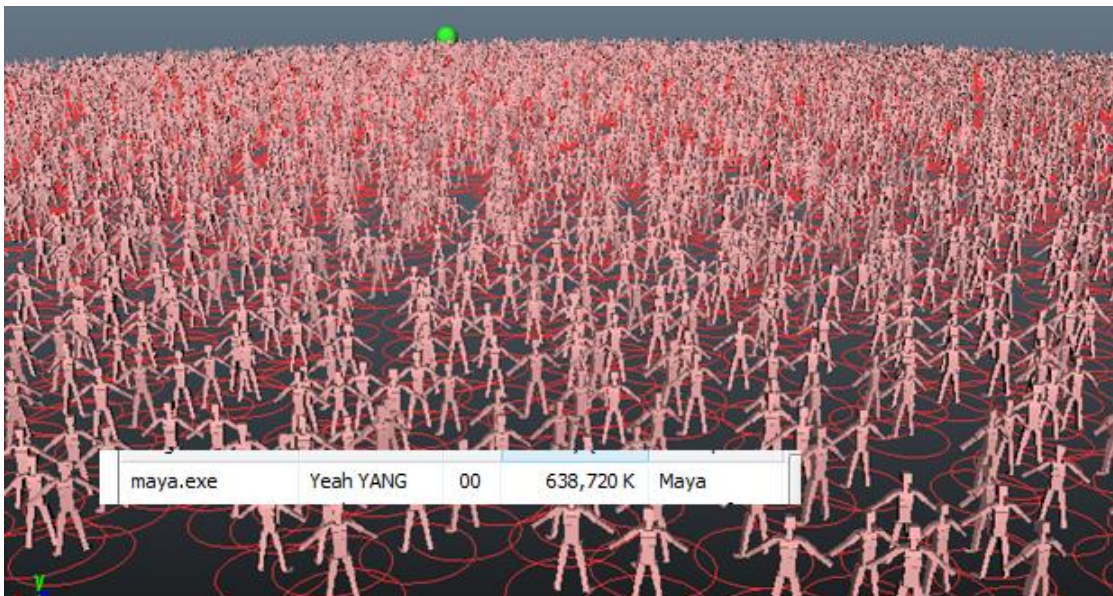


Follow spot by spot channels

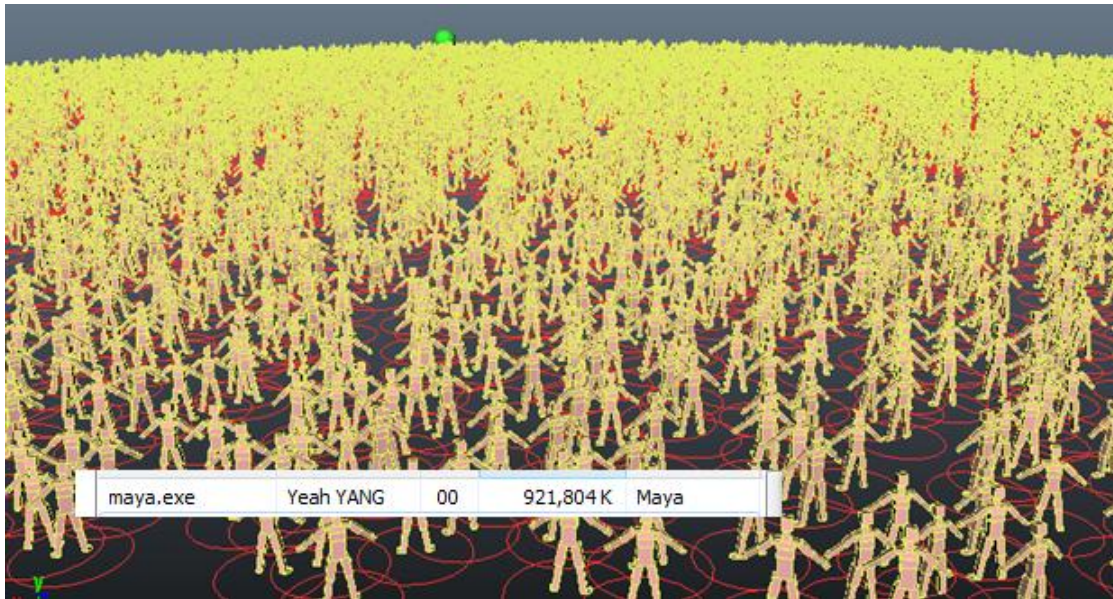
Future Solution: We are trying to use CUDA to make the calculation parallel in future, and we are still trying to using more robust agent filter feature to speed up the vision calculation.

Collide Family Channel & Pre-build RBD Object

All the collide-family channels (like “collide”, “collideAny” and so on) will make the system create pre-build RBD objects in agent memory because the collide-family channels are based on the PhysX RBD object.



5000 Agents in Maya by default (1.5 sec per frame calculate, sound + spot follow)



5000 Agents in maya with collide-family channels (about 3 extra seconds for updating kinematic RBD in agent memory);
Yellow boxes are the pre-build RBD display by PhysX Debug Node

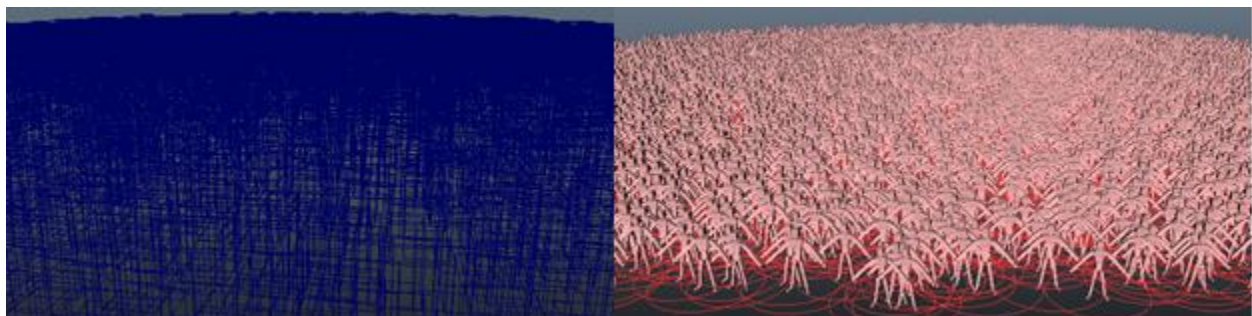
Collide Any Channel

The “collideAny” channel will check every bone of current agent, whereas the “collide” channel will only check the flagged bones. So, if the agent contains 100 bones and 10 of them are flagged, “collideAny” channel will spend 10 times of time than “collide” channel to calculate result.

“collideAt” channel has the fastest speed, because it only check one bone. “collideBy” channel has the same efficiency with the “collide” channel.

Bounding Box Display

Sometimes when there are too many agents your scene and the viewport is difficult to orbit, you can switch to bounding box display.

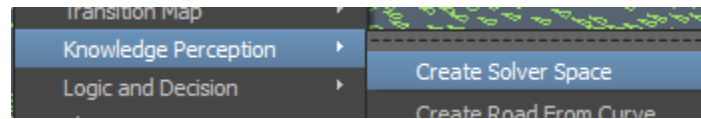


(Left) bounding box display (right) agent segment display

Solver Space

If you make sure the agents in different area or floors will not interactive each other forever, you'd better separate them in different solver space, it can save lot of time to calculate and easier to management.

If your agents are in different vehicles and the vehicles are moving each other, such like several battleships, you should group them in different solver space.



Create solver space and put place node inside

Easy Transition

Sometimes, using easy transition can same lot time for you.

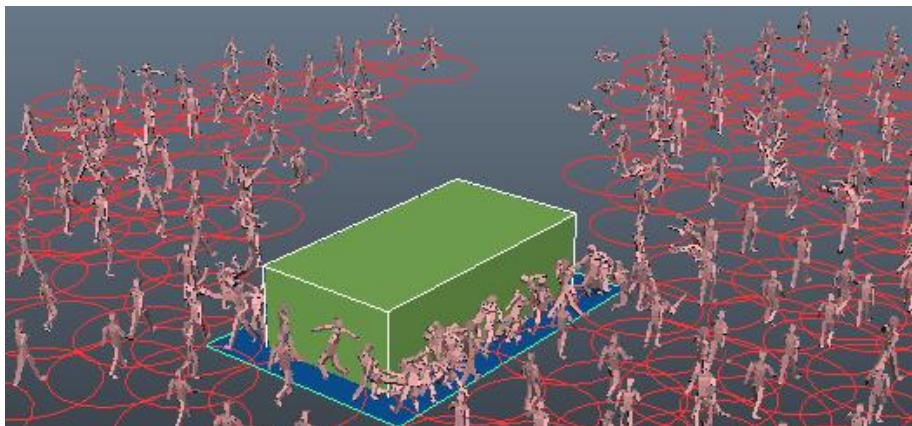
- Pre-vis
- Simple shot, agents have each solo action, factory operator, walking crowd, etc...
- Agents are far from camera will not been noticed by audience

Optimizing Crowd Dynamics

With properly setup, Miarmy is able to support more than 5000 agents enable dynamics at the same time.

Using Spot or Zone Trigger Dynamic

Sometimes we need use a kinematic object like some cars/rocks to collide crowd agents. We suggest that you use the zone/spot channels for activating dynamics instead of collide channels. Like the picture below, in the "car" scenario. The green kinematic primitive is a child of that blue zone geometry plane. We use the "zone.in" channel trigger dynamics active then the dynamics agents will naturally collide with the moving kinematic primitive.



Use "zone.in" channel to trigger dynamics

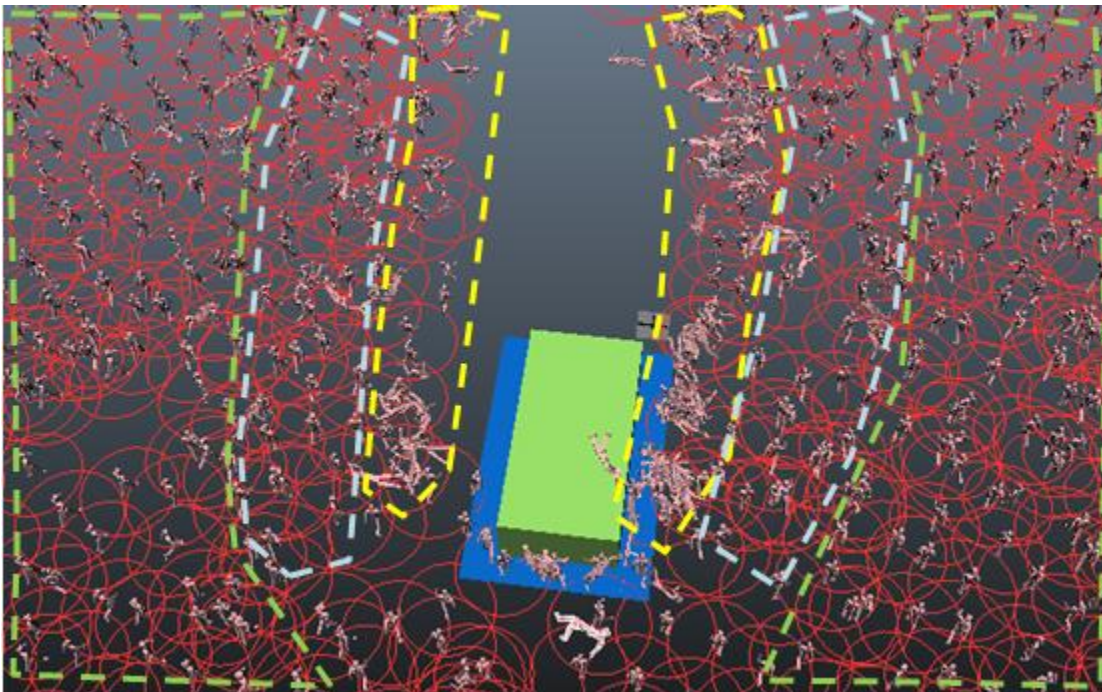
With the same theory, in the “rock” scenario, you need use “spot.d” or “bound.in” for triggering enable dynamics, and there is a sphere kinematic primitive been a child of bound node.



Use “bound.in” channel to trigger dynamics

Mute Dynamics

On each agent, there is a “muteDynamic” attribute on it. Once we turn this on, the agent will never enable dynamics and will not join the collision detection, the pre-build RBD object will never been created. In some cases, using this feature properly, it will save tremendously amount of memory for you as well as speed your simulation up.



Mute dynamics for the agent din blue bound and inverse placement

For example, in the above picture, the car (green kinematic primitive) is hitting into the crowd agents. Some agents (in the yellow bound range) will be enable dynamics and hit away. For improving the reality, we need check collision for the nearby agents (in the blue bound range) hitting by the coming dynamic agents. If the agents (in blue bound) hit by dynamic agents, the dynamics will be enabled. As we know, the agents will collide-family channels will created pre-build RBD Object in memory. However, there are still lots of agent we don't want them to become dynamic ones, because they are far from the accident. We want to disable dynamic them totally forever for saving memory and time. We need select them and enable "mute dynamic" attribute.

However, the attribute changes cannot be saved with scene, if the agents have been deleted and placed out again, these attribute will change back. So, you need enable "muteDynamics" for these agents first and use "inverse placement" to store these data to a new place node.

Part 10 Expanding Guide

This part is Undergoing, to be continued...

Scripts

Build your own logic Preset

Build your own channel Preset

Build your own automatic original agent setup pipeline

Build your own automatic action creation pipeline

Particle Express for Hybrid Dynamics Effects

API

Write your own Field Node

Render Fur

Use Miarmy Engine Drive Maya Character

Miarmy Reference

Callback Functions

After Load Miarmy

- After loading Miarmy, system will run a script jot which name is “McdInitMiarmy”. If you don’t want to automatically run this script, you need delete the contents from this file in you Miarmy installation place.

Before Save

- Before saving, system will unhide original agent group nodes and all place nodes.

After Save

- After saving, if agents have been placed out, system will hide original agent group nodes and all place nodes.

Selection Change

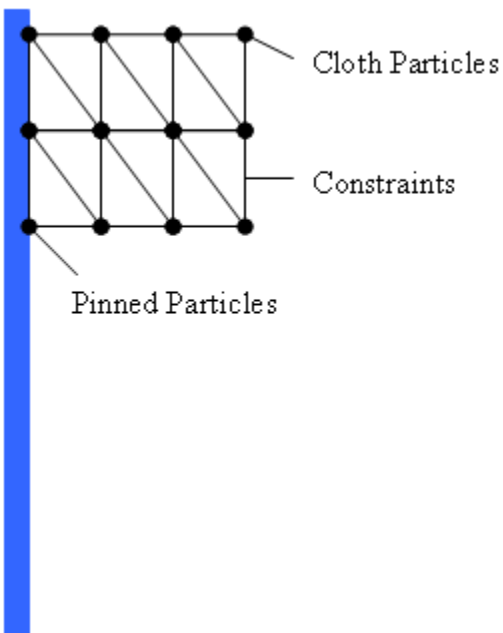
- Update python window for selected relative nodes, for example you are editing decision, and once the selected object changed, and if new select one is decision node ,the python GUI will be updated.
- If your new selected object is agent, system will mark feedback agent to engine and recalculate, then results will be put to Brain Viewer and Transition Map.

Before Duplicate

- If there is any agent, place and road node in your duplicate list, they will be unselected before duplicating and system will prompt out.

Cloth

The cloth feature of the NVIDIA PhysX SDK allows for simulation of items made from cloth, such as flags, clothing, etc. This is accomplished by providing a mesh that is used to define a set of particles (vertices). The topology of the mesh allows the SDK to construct constraints between the particles that mimic how cloth can bend and stretch. In addition, cloth particles can be pinned to shapes and global positions, such as attaching a flag to a pole.



There are two main types of cloth constraints:

- **Stretching** - applied to maintain distance between each particle that is connected by an edge in a cloth mesh. The strength of this constraint is assigned using the SDK; specifying a small stretching constraint factor allows the particles to move apart more easily and gives the impression of more stretchy cloth, such as lycra. Specifying a larger constraint factor makes the cloth stiffer, like denim.
- **Bending** - applied to maintain the angle along an edge in a cloth mesh, either by constraining the angle directly or by constraining the distance between the pair of particles on either side of the edge (see the diagram below). An example of a material that would use a low bending constraint for simulation is cotton, while a substance such as paper or cardboard would use a high bending constraint.

Channel Reference List

Input Channels

Sound: see details in Part 5 sound session

sound.x	relative directions to others agent in horizontal	[-180, 180]
sound.y	relative directions to others agent in vertical	[-180, 180]
sound.d	distances between others agent	[0, sound]
sound.ox	relative orientations to others agents in horizontal	[-180, 180]
sound.oy	relative orientations to others agents in vertical	[-180, 180]
sound.f	frequencies of agents in sound range	[0, inf)

Vision: see details in Part 5 vision session

vision.x	relative directions to others agent in horizontal	[-180, 180]
vision.y	relative directions to others agent in vertical	[-180, 180]
vision.z	distances between other agents (maya unit)	[0, inf)
vision.h	color ids of agents in vision range	[-1, inf)

Road (indexing enable) See details in Part 5 Road

road.flow	angle between agent Z-axis and the flow field	[-180, 180]
road.x	agent position in the road	[-1, 1]
road.ox	angle between agent Z-axis and the road	[-180, 180]
road[id].x	agent position in the road of which index is <id>	[-1, 1]
road[id].ox	angle between Z-axis and the road whose index is <id>	[-180, 180]

Terrain: See details in Part 5 Terrain

ground	height between the feet and the terrain	(-inf, inf)
ground.dx	relative angle between the terrain and agent X-axis	[-90, 90]
ground.dz	relative angle between the terrain and agent Z-axis	[-90, 90]

Bound (indexing enable)

bound.in	return 1 if agent in bound range	0 or 1
bound[id].in	return 1 if agent in bound range whose index is <id>	0 or 1

Spot (indexing enable)

spot.x	relative positions represented by angle in horizontal	[-180, 180]
spot.y	relative positions represented by angle in vertical	[-180, 180]
spot.d	distances with spots	[0, inf)
spot[id].x	same as spot.x, only feel spots whose index is <id>	[-180, 180]
spot[id].y	same as spot.y, only feel spots whose index is <id>	[-180, 180]
spot[id].d	same as spot.d, only feel spots whose index is <id>	[0, inf)

Zone (indexing enable)

zone.x	relative positions represented by angle in horizontal	[-180, 180]
zone.y	relative positions represented by angle in vertical	[-180, 180]
zone.in	whether in area of zone	0 or 1
zone.hi	height relative with zone	(-inf, inf)
zone.d3d	distances from agent to closest points	[0, inf)
zone.d (or .d2d)	distances from agent to closest points in XZ plane	[0, inf)
zone[id].x	same as zone.x, only can feel zone whose index is <id>	[-180, 180]
zone[id].y	same as zone.y, only can feel zone whose index is <id>	[-180, 180]
zone[id].in	same as zone.in, only can feel zone whose index is <id>	0 or 1
zone[id].hi	same as zone.hi, only can feel zone whose index is <id>	(-inf, inf)
zone[id].d3d	same as zone.d3d, only can feel zone whose index is <id>	[0, inf)
zone[id].d (or .d2d)	same as zone.d, only can feel zone whose index is <id>	[0, inf)

Wind (work with composition of force field)

wind.ox	orientaion relative with composition of wind in horizontal space	[-180, 180]
wind.oy	orientaion relative with composition of wind in vertical space	[-180, 180]
wind.a	amplitude of composition of wind	[0, inf)

Maya Field (work with composition of force field)

field.ox	orientaion relative with composition of fluid field in horizontal space	[-180, 180]
field.oy	orientaion relative with composition of fluid field in vertical space	[-180, 180]
field.a	amplitude of composition of fluid field	[0, inf)

Maya fluid (work with composition of force field)

fluid.a	amplitude of composition of fluid field	
fluid.ox	orientaion relative with composition of fluid field in horizontal space	[-180, 180]
fluid.oy	orientaion relative with composition of fluid field in vertical space	[-180, 180]

Collision Detection

riCollide	check collide for flagged bone (reserve info if collide)	0 or 1
riCollideAny	check collide for any bone (reserve info if collide)	0 or 1
riCollideAt:<bone>	check collide for bone whose name is <bone> (reserve info if collide)	0 or 1
riCollideBy:<bone>	check whether the flagged bones collide by bone whose name is <bone> (reserve info if collide)	0 or 1
riCollideAnyBy:<bone>	check whether any bone collide by bone whose name is <bone> (reserve info if collide)	0 or 1
collide	check collide for flagged bone	0 or 1
collideAny	check collide for any bone	0 or 1
collideAt:<bone>	check collide for bone whose name is <bone>	0 or 1
collideBy:<bone>	check whether the flagged bones collide by bone whose name is <bone>	0 or 1
collideAnyBy:<bone>	check whether any bone collide by bone whose name is <bone>	0 or 1

Self-Variable

hp	get hp value from agent memory	(-inf, inf)
mp	get hp value from agent memory	(-inf, inf)

Scene Info

frame	get current frame	[0, inf)
goFrame	get frame number since simulation start	[0, inf)

Noise

noise.id	return random value based on agent id	(0, 1)
noise.time	return random value based on current frame	(0, 1)

Action

<actionName>	return true if the current playback action is <actionName>	0 or 1
--------------	--	--------

Output Channels**Sound**

sound.f	set the frequency of agent sound	
sound.a	set the range of agent sound	

Vision

color	set color of agent (if -1, others cannot see it)	
-------	--	--

Move or Rotate Speed

tx	move speed of translate x, per second	(-inf, inf)
ty	move speed of translate y, per second	(-inf, inf)
tz	move speed of translate z, per second	(-inf, inf)
rx	change rate of roate x in degree, per second	(-inf, inf)
ry	change rate of roate y in degree, per second	(-inf, inf)
rz	change rate of roate z in degree, per second	(-inf, inf)

Aiming

<bone>:aim<axis><space>(<percent>)-><target>	percent aim to target, see aiming session	(0, 1)
<bone>:aim<axis><space>-><target>	100% aim to target, see aiming session	(0, 1)

Self-Variable

hp	set hp value or hp change rate (in change rate mode)	(-inf, inf)
mp	set mp value or mp change rate (in change rate mode)	(-inf, inf)
hp.set	set hp value (in both mode)	(-inf, inf)
mp.set	set mp value (in both mode)	(-inf, inf)

DDIM 1.0 (Dynamically Decrease Intelligence Mechanism 1.0)

vision.mute	disable vision for itself and also let others cannot see this agent	[1]
sound.mute	disable sound for itself and also let others cannot feel this agent by sound	[1]

Ragdoll

dynamics.active	enable full body ragdoll	[1]
dynamics.active.force	enable full body ragdoll and apply a pulse force from collide reserved info	[0, inf)
bodyDynamics.active	enable body dynamics with root bone kinematic control	[1]
<boneName>:dynamics.active	enable dynamic from boneName	[1]
<boneName>:dynamics.detach	enable dynamic from boneName and break joint	[1]

Action

<actionName>	trigger action	[0, inf)
<actionName>:rate	modify action rate if this action playing back	[0, inf)
<actionName>-><otherActionName>	modify action blend if this action playing back	[0, 1]
actionGroup:<actionGrouName>	trigger one of action in action group randomly	[0, inf)

Bone Offset

<boneName>:tx	set the translate X in bone offset	(-inf, inf)
<boneName>:ty	set the translate Y in bone offset	(-inf, inf)
<boneName>:tz	set the translate Z in bone offset	(-inf, inf)
<boneName>:rx	set the rotate X in bone offset	(-inf, inf)
<boneName>:ry	set the rotate Y in bone offset	(-inf, inf)
<boneName>:rz	set the rotate Z in bone offset	(-inf, inf)