ChAOS

Crowd Animation Open Software

User’s documentation

JANUARY 8, 2019

INRIA

Table of Contents

1. Overview

a. Introduction

ChAOS is a visualization tool to create a crowd with animated virtual humans from trajectory files. It can be used to visualize the results of a crowd simulation, replay data from tracked individuals or simply having animated virtual humans followings specific trajectories. ChAOS is able to save pictures at a specific framerate, which can be used to generate video with other tools such as ffmpeg.

b. Running ChAOS

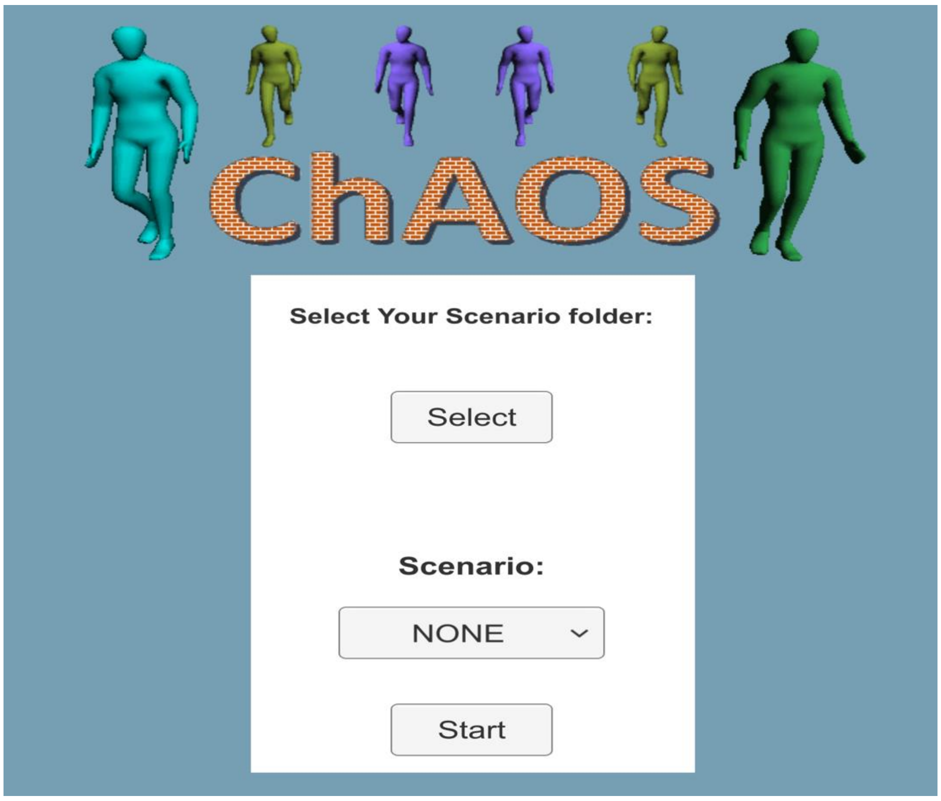


Figure 1: ChAOS starting menu

Trajectories can easily be animated using ChAOS. First the trajectories need to be translated in the right format, described in section 3.b. Then a scenario file, loading the trajectories and setting the animation parameters, has to be created (see section 3.a). Once this is done, the ChAOS executable can be run. In the application menu, shown on Figure 1, you have to select the scenario folder. Then select your scenario and click the start button to load it. Finally, hit the space key from the keyboard or press the "play button" on the screen to start the animation.

To create a video, modify the configuration file to enable recording (see explanation page 3) then launch the executable. Access to the configuration panel by hitting the "p" key from the keyboard or select the "Setting" menu on the top of the screen. Once the recording is over, the application should quit automatically and a set of pictures should be present in the output directory specify in the configuration file. A video can be generated from this set of pictures using a third-party software such as ffmpeg. Here is an example of command to generate a video with ffmpeg ("-r 15" define the input framerate as 15 and should be change to correspond to the recording framerate specify in the ChAOS configuration file):

```
ffmpeg -r 15 -i img%03d.png -c:v libx264 -vf fps=25 -pix_fmt yuv420p out.mp4
```

2. Controls

a. General commands

Key	Description
escape	Open/Close the menu.
Space	Pause/Resume the animation
P	Access to the configuration panel

b. Camera movement

Key	Description
Arrows	Translate the camera in the direction of the arrows (forward, backward, right and left).
Shift + Arrows	Accelerate the camera translation induce by the arrows.
Ctrl + Arrows	Translate the camera in the direction of the arrows on the 2d plane independent from elevation (forward, backward, right and left).
Mouse right click	Rotate the camera following the mouse movement.
F5	Save the camera positioning in the scenario file to reuse it in future replay.

c. Camera filters

Key	Description
F9	Switch to the previous camera filter
F10	Switch to the next camera filter
F11	Reset filter to original image

3. Files format

a. Scenario files

```
<?xml version="1.0" encoding="utf-8"?>
<ConfigData xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <!-- Specify the directory with all the trajectory files -->
  <env_filesPath>.\Scenarios\ExampleTwoColors\</env_filesPath>
  <env_obstFile>.\Scenarios\Decor\obstExampleTwoColors.xml</env_obstFile>

  <!-- Camera positioning -->
  <cam>
    <cameraType typeID="3" />
    <position x="0" y="0" z="30" />
    <rotation x="90" y="0" z="0" />
    <lookAtAgent agentID="2" />
    <followAgent agentID="2"
                 followOnX="true"
                 followOnY="true"
                 lockFirstPerson="false"
                 smoothFirstPerson="true" />
    <CamResolution x="-1" y="-1" />
  </cam>

  <!-- Configure recording: save picture in the give directory from start
to end with specific framerate -->
  <recording start="0" end="0" framerate="15" width="-1" height="-1">
    <saveDir>.\Output\</saveDir>
    <saveImgOriginal record="true" quality="8"
                     width="-1" height="-1" />
    <saveImgSegmentation record="false" quality="8"
                         width="-1" height="-1" />
    <saveImgCategories record="false" quality="8"
                       width="-1" height="-1" />
    <saveImgDepth record="false" quality="8"
                  width="-1" height="-1"
                  minDepth="0" maxDepth="50" exponent="1" />
    <saveImgNormals record="false" quality="8"
                    width="-1" height="-1" />
    <saveImgOpticalFlow record="false" quality="8"
                        width="-1" height="-1"
                        motionVector="false" />
    <saveBodyBoundingBoxes record="false" />
    <saveHeadBoundingBoxes record="false" />
  </recording>

  <!-- Define the colors of the agents (agents not in the list get a
random colors) -->
  <AgentColorList>
    <color firstAgent="0" step="1" lastAgent="191" red="1" green="0"
blue="0" />
    <color firstAgent="192" step="1" lastAgent="383" red="0" green="1"
blue="0" />
  </AgentColorList>

</ConfigData>
```

Figure 2: Scenario file example

A scenario is composed of a set of trajectories to animate and a few parameters that can be optional or mandatory. The scenario parameters are described in an xml file, an example of such file is given in Figure 2. The list of parameters that can be set by the scenario file is given in the Tableau 1.

XML parent	Parameter	Description
	env_filesPath	The path to the directory containing all the

		trajectory files.
	env_obstFile	The path to the XML files containing a list of obstacles
	env_stageInfos	Load the gameObject (scenery) named *stageName* from a unity asset bundle file
env_stageInfos	*file*	The path to the unity asset bundle file
env_stageInfos	*position*	Position of the loaded gameObject
env_stageInfos	*rotation*	Rotation of the loaded gameObject
	cam	List the parameters specific to the camera. (The position and rotation can be set during runtime and saved using the F5 key.)
Cam	*cameraType*	The different types for the camera defined by *typeID* correspond to the following: (**0**)*Free*: free movments. (**1**)*Follow*: the camera follow the *agentID* of followAgent. (**2**)*LookAt*: the camera point at the *agentID* defined by *lookAtAgent*. (**3**)*FirstPerson*: the camera is attached to the user *agentID* of *followAgent*. (**4**)*Torsum*: similar to *FirstPerson*, but attached to the torsum. Check section 7 for more details.
Cam	*position*	Initial Position of the camera
Cam	*rotation*	Initial Rotation of the camera
Cam	*lookAtAgent*	In (**2**)*LookAt* mode it defines the agent *agentID* in sight, constraining the camera rotations.
Cam	*followAgent*	In (**1**)*Follow*,(**3**)*FirstPerson* and (**4**)*Torsum* modes it defines the agent *agentID* to follow, constraining the camera translations. In (**1**) mode: If *followOnX*="true" enables translation on X. If *followOnY*="true" enables translation on Y. In (**3**),(**4**) modes: If *lockFirstPerson*="true" the camera rotations are constrained to look ahead of the agent. If *smoothFirstPerson*="true" the oscillating movement of the camera are smoothed out.
	recording	Activate recording between the time *start* and *end* with specific *framerate*. Recorded image resolution can be forced by specifying the *width* and *height* (set to -1 to use the windows resolution).
Recording	*saveDir*	The path to the directory where all the recording pictures are saved
Recording	*saveImg[Type]*	Define a type of image to save with the following parameters: The Boolean *record* whether the image is recorded.

		quality which can be 8, 16 or 32 defines the number of bit per channel use during recording. *width* and *height* define the resolution of the recorded image (overwrite the parameters from the parent object) Specific parameters see section 5.a for more details.
Recording	*saveBodyBoun-dingBoxes* *saveHeadBoun-dingBoxes*	Define whether the body and/or the head bounding boxes data are saved through the Boolean *record*
	AgentColorList	List the color for all the agents with specific colors
AgentColorList	*color*	Specify a color for every *step* agents from *firstAgent* to *LastAgent* The color is set from and RGB code with *red*, *green* and *blue* value between 0 and 1 If the color of an agent is not specify, it is randomize.

Tableau 1: The list of XML items uses by the scenario file

b. Trajectory files

```
0.1,-21.9582,-0.429098
0.2,-21.8447,-0.429502
0.3,-21.7325,-0.434389
0.4,-21.6205,-0.440434
0.5,-21.5083,-0.446389
0.6,-21.3959,-0.451533
0.7,-21.2831,-0.455441
0.8,-21.1699,-0.457503
0.9,-21.0565,-0.458042
1.0,-20.9430,-0.458283
1.1,-20.8331,-0.453430
1.2,-20.7230,-0.448326
1.3,-20.6130,-0.443207
1.4,-20.5032,-0.441068
1.5,-20.3935,-0.438831
1.6,-20.2841,-0.436204
1.7,-20.1748,-0.433453
1.8,-20.0656,-0.430283
1.9,-19.9565,-0.426964
2.0,-19.8475,-0.423200
```

Figure 3: Trajectory file example (Format: time, x, y)

Trajectory files are a set of CSV files, each containing the trajectory of one agent. The files are sorted alphabetically and the ID of the agent correspond to its position in the sorted list, starting from 0. A trajectory file follows the following format: “Time, position.X, position.Y” as shown by the example in Figure 3.

c. Obstacles files

```xml
<?xml version="1.0" encoding="utf-8"?>
<XMLObstacles xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <!-- Define Rectangular shaped obstacles -->
  <!--          a line with a width         -->
  <Rectangles>
    <Rectangle Width="0.2">
      <A X="-50" Y="-3" />
      <B X="50" Y="-3" />
    </Rectangle>
    <Rectangle Width="0.2">
      <A X="-50" Y="3" />
      <B X="50" Y="3" />
    </Rectangle>
  </Rectangles>

  <!-- Define Cylindrical shaped obstacles -->
  <!--         a point with a radius        -->
  <Cylinders>
    <Cylinder Radius="1">
      <Center X="2.5" Y="0" />
    </Cylinder>
    <Cylinder Radius="2">
      <Center X="5" Y="1" />
    </Cylinder>
  </Cylinders>

</XMLObstacles>
```

Figure 4: Obstacle file example

When loading a set of trajectories, the animation is play in an empty environment. ChAOS enables you to add simple obstacles in the environment (rectangular and cylinder shaped). The obstacles' parameters are describe in an xml file that is loaded by the scenario file using the env_obstFile parameter (see section 3.a). An example of obstacles file is show on Figure 4.

XML parent	Parameter	Description
	Rectangles	List of rectangle shaped obstacles
Rectangles	*Rectangle*	Define a rectangle shaped obstacle: Create the line *AB* with a specific *Width*
Rectangle	*A*	Position of the first point of the line
Rectangle	*B*	Position of the second point of the line
	Cylinders	List of cylinder shaped obstacles
Cylinders	*Cylinder*	Define a cylinder shaped obstacle: Create a cylinder from a *center* and a *radius*
Cylinder	*Center*	Position of the center of the cylinder

d. Stage files

For more complex stage with specific structures, ChAOS is able to load Unity's asset using the AssetBundle Workflow. First, the stage needs to be design in Unity. Then the AssetBundle can be built (see Unity doc: https://docs.unity3d.com/Manual/AssetBundles-Workflow.html). Finally, to load the AssetBundle in ChAOS, its path has to be specify in the scenario file using the env_stageInfos parameter (see section 3.a).

4. Camera Settings

There are different available parametrization for the camera, to allows modification to the orientaiton and position of the camera.

a. Camera modes

We have defined 4 different modes:

0. **Free:** move free in the environment starting from the current position (reset ->restore default position).
1. **Follow:** the camera position follows the agent defined in *AgentID* (if not defined it is automatically set to the id 0) on the axes selected with *followOnX* and *followOnY.*
   - Additional parameters:
     - **a.** *AgentId :* the id of the agent to follow.
     - **b.** *followOnX :* enable translation on X.
     - **c.** *followOnY* : enable translation on Y.
   - **Movement:**
     - **a.** Translation: free with automatic translation.
     - **b.** Rotation: free.
   - Reset button: restore default orientation.
2. **Look At:** the camera orientation follow the agent defined in *AgentID* (if not defined is automatically set to the id 0).
   - Additional parameters:
     - **a.** *AgentId :* the id of the agent to look at.
   - **Movement:**
     - **a.** Translation: free on the X and Y axes.
     - **b.** Rotation: constrained.
   - Reset button: restore default position.
3. **FirstPerson:** It attaches the camera to the head of the user.
   - Additional parameters:
     - **a.** *smoothFirstPersonView :* smooth out the oscillations of the camera.
     - **b.** *lockFirstPersonView :* lock the rotation of the camera.
   - **Movement:**
     - **a.** Translation: constrained.
     - **b.** Rotation: free/constrained.
   - Reset button: set the orientation of the camera to look in front.
4. **Torsum:** It attaches the camera to the torsum of the user.
   - Additional parameters:
     - **a.** *smoothFirstPersonView :* smooth out the oscillations of the camera.
     - **b.** *lockFirstPersonView :* lock the rotation of the camera.
   - **Movement:**
     - **a.** Translation: constrained.
     - **b.** Rotation: free/constrained.
   - Reset button: set the orientation of the camera to look in front.

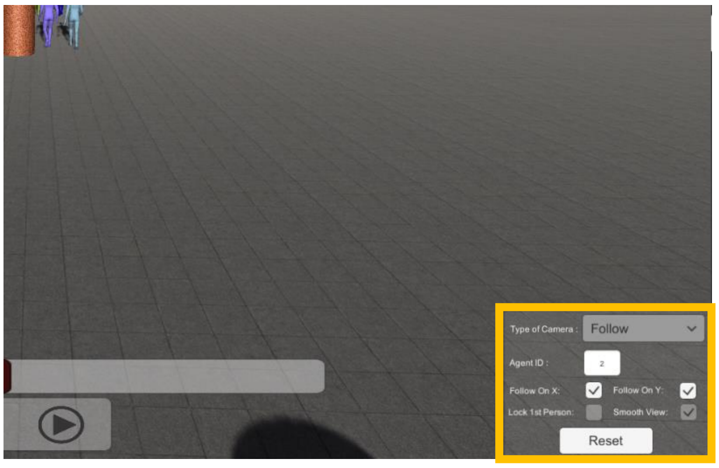
b. Config of camera Settings

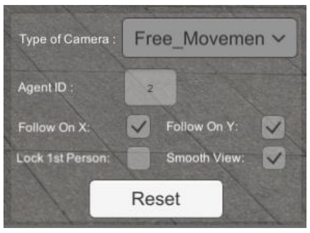
In the configration file all the parameters are exposed and the mode can be selected by the *typeId* of the *cameraType* tag, using the id from 0 to 4, this corresponde to the modes lisited in the previous section.

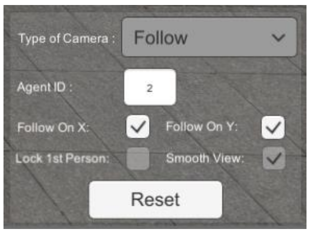
```
...
  <!-- Camera positioning -->
  <cam>
    <cameraType typeID="3" />
    <position x="0" y="0" z="30" />
    <rotation x="90" y="0" z="0" />
    <lookAtAgent agentID="2" />
    <followAgent agentID="2"
                        followOnX="true"
                        followOnY="true"
                        lockFirstPerson="false"
                        smoothFirstPerson="true"
               />
    <CamResolution x="-1" y="-1" />
  </cam>
...
```

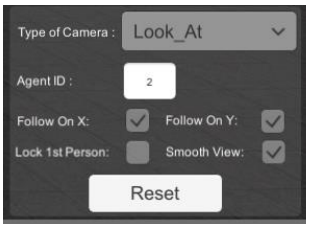
a. Interface of camera Settings

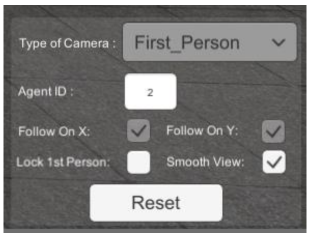
The interface to access the camera parameters is on the bottom right part of the screen .

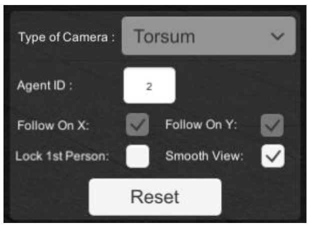

Type of Camera : Follow
Agent ID : 2
Follow On X: ✓ Follow On Y: ✓
Lock 1st Person: Smooth View: ✓


Type of Camera : Free_Movemen
Agent ID : 2
Follow On X: ✓ Follow On Y: ✓
Lock 1st Person: Smooth View: ✓


Type of Camera : Follow
Agent ID : 2
Follow On X: ✓ Follow On Y: ✓
Lock 1st Person: Smooth View: ✓


Type of Camera : Look_At
Agent ID : 2
Follow On X: ✓ Follow On Y: ✓
Lock 1st Person: Smooth View: ✓


Type of Camera : First_Person
Agent ID : 2
Follow On X: ✓ Follow On Y: ✓
Lock 1st Person: Smooth View: ✓


Type of Camera : Torsum
Agent ID : 2
Follow On X: ✓ Follow On Y: ✓
Lock 1st Person: Smooth View: ✓

The buttons are automatically activated if we are currently in the related mode.

5. Learning

Chaos is able to produce data that can be relevant for learning algorithm. You will find a description of these data in this section.

a. Camera filters

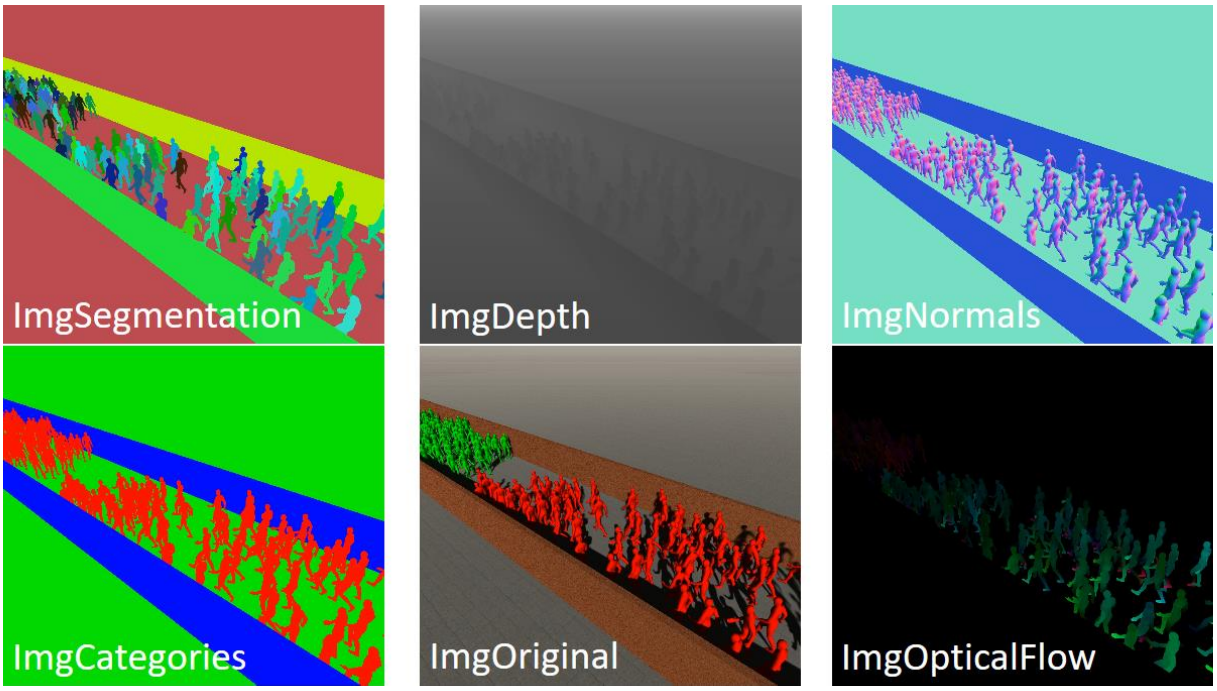


Figure 5: Available camera filters.

The first type of data generated for learning is images. From the original images used to create videos of crowd trajectories with animated virtual humans, 5 filters can be applied to store different data on the pixels of the images (see Figure 5):

- **ImgSegmentation**: this filter will color every objects in the image with a different color segmenting every virtual humans as well as every parts of the stage.
- **ImgCategories**: This filter will color every objects in the image with a color corresponding to its type: red for virtual humans, blue for obstacles and green for the ground. For personalized stage file import via AssetBundle, object as to be categorized manually by selecting a specific unity layer for every GameObject. Layer 8 is for virtual humans, layer 9 is for the ground, layer 10 is for obstacles and any layer above can be used for other types of object.
- **ImgDepth**: This filter color every pixel according to the distance from the camera of the corresponding object point. The distance (*dist*) is transform to a shade of grey (*shade*) using the following equations (red variables defined in the scenario file see section 3.a):

$$shade = \begin{cases} 0 \; if \; dist < MinDist \\ \left(\frac{(dist - MinDist)}{(MastDist - MinDist)}\right)^{exponent} \; if \; MinDist < dist < MaxDist \\ 1 \; if \; dist > MaxDist \end{cases}$$

- **ImgNormals**: This filter color every pixel according to the normal of the corresponding surface.
- **ImgOpticalFlow**: This filter color every pixel according to the direction of movement of the corresponding object point. The color wheel used by direction is represented by the Figure 6. The vector optical flow (Vx, Vy) can be recorded instead by setting the

parameter motionVector to true (see section 3.a) in which case the R channel will be Vx and the G channel will be Vy.

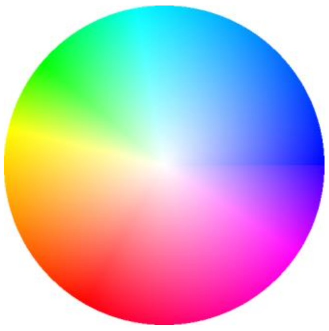
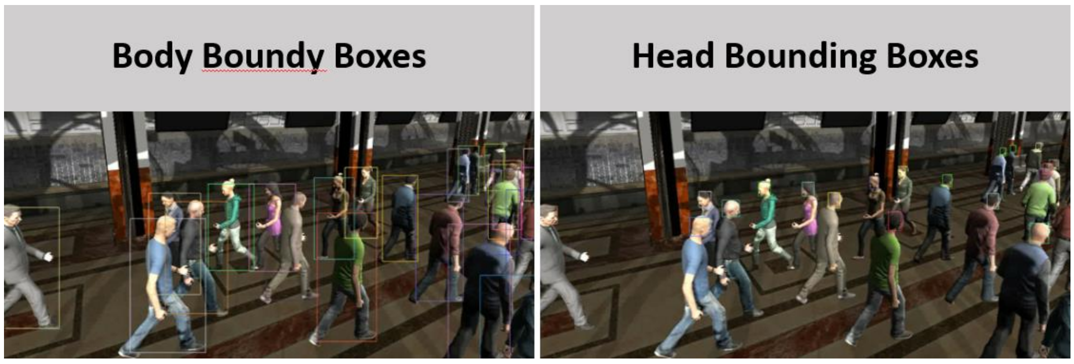


Figure 6: Optical Flow Wheel

b. Data



The data generated by this module will be a csv file for each recorder (body or head bounding box). The architecture of the ground truth is inspired by the MOT challenge. In this file you will find the ground truth for each object at each frame with a specific sequences of values. Each line of the file will contain 10 values:

```
<frame>, <id>, <bb_left>, <bb_top>, <bb_width>, <bb_height>, <conf>, <x>, <y>, <z>
```

With :

- Frame : number of the current frame
- Id : id of the agent
- bb_left, bb_top, bb_width, bb_height: bounding box information
- conf : confidence detection in our case 1
- x,y,z : the object world coordinates (for now -1,-1,-1)

6. Command Line

It is also possible to launch Chaos via a command line by giving several input parameters. You will find below the list of parameters to use:

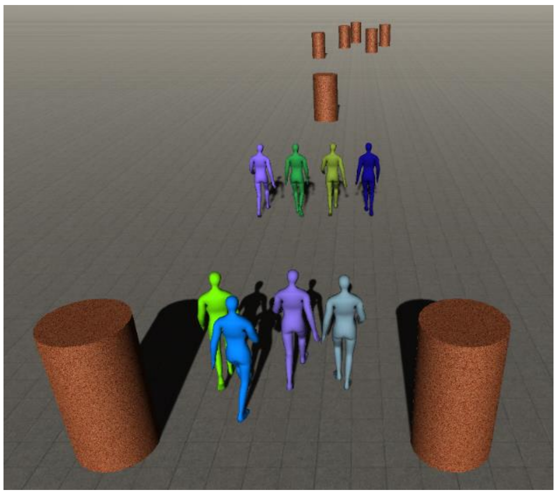
- `-s PathScenario`: Indicates the path (*PathScenario*) to the scenario you want to run.
- `-r Width Height`: Indicates the resolution (*Widht,Height*) you want to use for recording the image.
- `-batchmode`: Indicates that Chaos will run without graphic window

This is an example of using Chaos via command line on windows: `ChAOS.exe -s Scenarios\ExampleLookAt.xml -r 720 480 -batchmode`

7. Examples

The application is provided with three scenarios as examples covering the main features of ChAOS. These scenarios are in the default scenarios folder and are selectable directly from the starting menu. To run one, select it on the starting menu and click run. The three scenarios are described below:

ExampleFollow



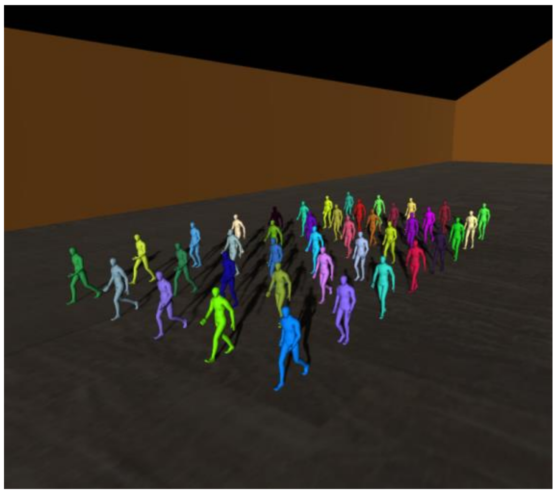
Features:

- Camera following an agent
- Creation of cylinders shaped obstacles
- Agents spawning after the start of the animation
- Agents disappearing before the end of the animations

List of files (in the Scenarios folder):

- Scenario file (ExampleFollow.XML)
- Trajectory folder (ExampleFollow/)
- Obstacle file (Decor/obstExampleFollow.xml)

ExampleLookAt



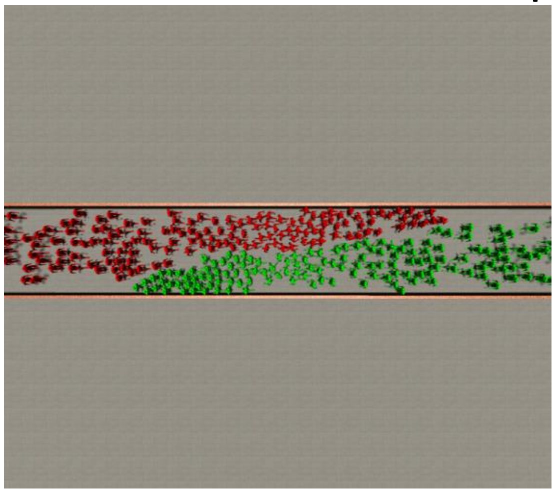
Features:

- Camera looking at an agent
- Loading a stage from an AssetBundle

List of files (in the Scenarios folder):

- Scenario file (ExampleLookAt.XML)
- Trajectory folder (ExampleLookAt/)
- AssetBundle (Decor/building)

ExampleTwoColors



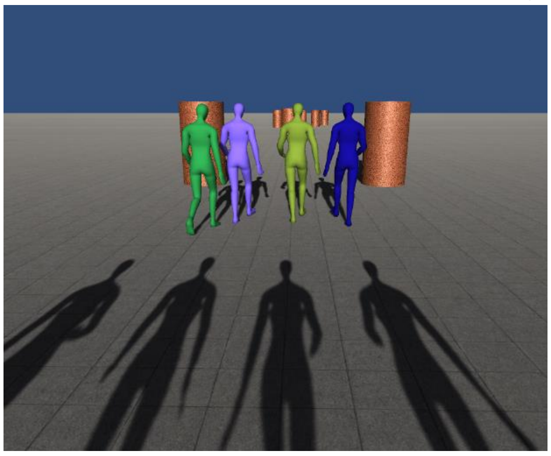
Features:

- Creation of rectangle shaped obstacles
- Forced agents' color

List of files (in the Scenarios folder):

- Scenario file (ExampleTwoColors.XML)
- Trajectory folder (ExampleTwoColors/)
- Obstacle file (Decor/obstExampleTwoColors.xml)

ExampleFirstPerson



Features:

- Camera in the head of the agent
- Creation of cylinders shaped obstacles
- Agents spawning after the start of the animation
- Agents disappearing before the end of the animations

List of files (in the Scenarios folder):

- Scenario file (ExampleFollow.XML)
- Trajectory folder (ExampleFollow/)
- Obstacle file (Decor/obstExampleFollow.xml)