Double Degree Program Thesis

(Politecnico di Torino)

Development of Artificial Intelligence-Based Tools for Houdini Software for Static and Dynamic Crowd Replications

(Desarrollo de Herramientas basadas en algoritmos de inteligencia artificial para el software Houdini para simulación de multitudes)

Tutors:

Chiara Boeri (Politecnico Di Torino)
Stefano Leoni (Effetti Digitali Italiani Edi)

Candidate:

Rafael Arturo Rey Camacho

October, 2010
Summary

Actually the world is suffering the exponential growth of the computers capabilities and the great increase of their possible applications. The entertainment industry requires more and more solutions in a fast way with the best quality to catch the attention of the public and obtain benefits.

In the last years could be seen the development of new technologies and the improvements of the digital solution for cinema and television. With this evolution of the video processing, the visual effects that can be actually found on the movies are more impressive and complex, making the spectators hard to impress and making harder to catch their attention.

This fact has forced to the film makers to create new techniques and effects in order to generate curiosity and expectancy between the final recipients. This is the reason to find on the market the 3D films boom and the notable increment of 3D animation movies productions. That great development of new techniques for 3D animations has opened a new window: the creation of new features mixing those techniques to achieve spectacular and unique results.

The Italian market is not an exception and requires intelligent solutions to fill the continuous demand. This project has been developed inside one of the most important visual effects company in Italy (EDI) where the development of new features is fundamental to remain great positioned on the Italian and European market. This professional environment provides the necessary support and experience to create a high level and reusable tools.

This project takes most of the more important and powerful animation techniques and combines it with learning and knowledge theories and Artificial Intelligence programming to develop human-like crowd simulation tools for static and dynamic situations. Within the 3D animation techniques that have been utilized can be found Motion Capture with a huge library provided by the Carnegie Mellon University; key framing animation to set up different scenes; behavioral and geometric modeling to place the agents in the scene and define their actions; geometrical rigging to animate the Human-like geometries with the skeletons provided by the motion capture; and computer resources management to calculate the convenient Level Of Detail for each situation. All of this inside the sidefx’s Houdini software which license and apprentice have been provided by EDI.

The basis of the project, beyond the 3D animation techniques, is the implementation of some algorithms of one of the most important and interdisciplinary argument: Artificial Intelligence. The agents take the information of their environment to produce the best response for the situation, using a previous description of the world in the Houdini’s programming language (HScript). Also as guidance are taken algorithms of Artificial Intelligence for video games in order to implement collision avoidance, obstacles avoidance, follow terrain, etc.

All the user interface is developed in Houdini with a powerful tool provided inside it: The digital Asset design. Inside this digital asset design environment can be mixed programming with scripting, using object oriented languages such as Hscript and Python. Also the programming paradigm used to develop independent agent in a complete guided animation environment, is different from the conventional programming paradigm. Here the variables must be created from the attributes that can be in vertex level, point level, primitive level or detail level. Then those attributes with a variable mapping can be read from other subnets, SOPs or nodes inside the Houdini’s environment. This programming paradigm permits to develop the AI algorithms in a 3D animation context.

The final result shows some tools that are able to create complex animations by just tuning different parameters. Those parameters are carefully selected to produce different outcomes without modifying the basis of the algorithms. On this document is explained how the theory is applied and the way the tools work to reach the complete creation of crowd replication animations.
## Contents

1. **Introduction** ........................................................................................................... 5

2. **Learning and Knowledge** ....................................................................................... 7

   2.1 **The Cognitive Approach** .................................................................................. 7

      2.1.1 Levels of analysis ......................................................................................... 7

      2.1.2 Cognitive Approach on Artificial Intelligence ................................................ 7

      2.1.3 Cognitive Architectures ................................................................................. 7

      2.1.4 Intelligent Agents .......................................................................................... 7

   2.2 **Artificial Intelligence** ......................................................................................... 8

      2.2.1 What is Artificial Intelligence? ....................................................................... 8

      2.2.2 Games and Animation AI .............................................................................. 8

      2.2.3 The kind of AI ............................................................................................... 8

      2.2.4 Movement ..................................................................................................... 9

      2.2.5 Steering Behaviors ........................................................................................ 10

      2.2.6 Decision Making ......................................................................................... 11

      2.2.7 Artificial neural networks .......................................................................... 12

3. **3D Animation** ......................................................................................................... 13

   3.1 **Motion Capture** ............................................................................................... 13

      3.1.1 Terminology and Notation ............................................................................ 13

      3.1.2 Methods and Systems ................................................................................... 14

      3.1.3 Motion Capture Database – Carnegie Mellon University ............................ 15

      3.1.4 File Formats .................................................................................................. 17

   3.2 **Computer Architecture for Virtual Reality** ......................................................... 19

      3.2.1 Stages ........................................................................................................... 20

      3.2.2 Graphic Station Architecture ....................................................................... 21

3.3 **Modeling** ............................................................................................................ 22

      3.3.1 Geometric Modeling .................................................................................... 22

      3.3.2 Cinematic Modeling ..................................................................................... 22

      3.3.3 Behavior modeling ...................................................................................... 23

4. **Crowds Simulation Project** ....................................................................................... 24

   4.1 **Overview** ........................................................................................................ 24

   4.2 **General Description** ...................................................................................... 25

   4.3 **About EDI** ..................................................................................................... 26

5. **Casting.otl** ............................................................................................................. 26

   5.1 **Introduction** .................................................................................................... 26

   5.2 **General Description** ...................................................................................... 27

   5.3 **How to use** ..................................................................................................... 27

      5.3.1 Inputs ............................................................................................................ 27

      5.3.2 What is displayed? ...................................................................................... 28

      5.3.3 Uses, Limitations and Possible Modifications .............................................. 28
1 INTRODUCTION

From ancient times the human beings have been trying to create human like objects in order to supply difficult and time consuming tasks. All this development arrived to a point where scientists and philosophers try to explain the way the human brain works: how it learns, how it stores the knowledge and how it uses the acquired knowledge to use it in an appropriate situation.

With the pass of time, different approaches have emerged trying to explain the process that suffers the knowledge inside the brain. Theories like behaviorism, cognitivism and constructivism explain this process in really different ways putting their force point in different parts of the process that depending on the time and the situation were considered the most important. Behaviorism shows the brain as a black box where all the actions are reactions to an external stimulus. Constructivism tells that all the knowledge is acquired by the experiences and past actions. The approach that is well utilized in engineering and artificial intelligence is the cognitive approach.

The cognitive approach explains the brain in a more complex and abstract way. The brain is no longer a black box and became a complex system that processes the information. The process is divided in level of analysis depending on the final goal of the specific situation. Marr divides the process in three big levels: Computational theory, Representation and algorithm, Hardware implementation. This approach is highly used in the Artificial Intelligence where some human aspects are implemented in a machine to complete an objective. Inside this context emerges the concept of intelligent agents. Russell and Norving provides on their book “Artificial Intelligence: A Modern Approach”, a great explanation of what are intelligent agents and how can be defined and classified within different contexts.

This framework conduces to one of the biggest interdisciplinary topics in the world: Artificial Intelligence (AI). Its basic concepts and definitions are explained from different fields like philosophy, academic and engineering and its historical evolution to fill human tasks have changed the world in the last century. Some different approaches also emerge to simulate artificial intelligence, between the most important and used are Hacks, Heuristics and Algorithms. Hacks are tightly related to the behaviorism and just take the reactions and implements the easier way to reproduce it. Heuristics that are also used in electronic controls are based in the error correction. It tries and corrects the mistakes. The last approach is algorithms. Algorithms provide a reusable tool to different situations. It is nested with the cognitive approach.

Nevertheless the artificial intelligence takes a great action field and for the purpose of this project the attention will be centered in just one: the artificial intelligence for animated agents using algorithms.

For this reason is considered the game AI as the center of the development. Within this context the artificial intelligence is divided in three stages: Movement, Decision Making and Strategy.

The movement stage is where the agents are recreated in a 3D context and set different aspect in order to achieve a realistic movement of the agents. Here are taken in consideration to the implementation of algorithms the linear transformation of objects in a coordinate system, the static simulation and the kinematic simulation. Those algorithms are adaptable to produce different outcomes and behaviors for a 3D character by creating and matching variables, such as obstacles avoidance, separation and avoidance (between agents), path following, etc.

The second stage is the decision making stage. Here the variables created by the movement are stored and processed in order to produce the appropriate response for the situation. Many algorithms for decision making have been developed, some more complex like neural networks, others that can be adaptable depending on the context to achieve a fast and easy outcome like decision trees or state machines. Some other algorithms that are used in the entertainment industries such as fuzzy logic (decision making used in Massive software) and rule based decision (L-systems modeling) serves to understand the actual state of the industry and know a little bit about its future.

The third stage, that is the strategies’ implementation is not taken into account for the purpose of this project, nevertheless it represents how a virtual character rules its behavior in order to achieve a single goal.
The interdisciplinary nature of AI permits and forces to locate this framework inside an application context. This project involves the creation of AI-based tools for 3D animation, so the next chapter involves the concepts and techniques used in 3D animation.

To create an AI animation is necessary to have realistic animations. The conventional animation techniques don’t provide an efficient way to reproduce some things such as the human movements or the fluids simulation. The motion capture technique brings the opportunity to have realistic animations by capturing it from a real human being. This technique is not yet standardized and there are too many ways to acquire the information. Can be used passive or active markers, can be optical, mechanical or magnetic systems. This fact have produced that many companies have developed their own system. Each system provides its own way to capture and package the data. During the creation of the AI tools is used a very large library of movements provided by the Carnegie Mellon University. This library has around 2500 different human-like movements and is free to use by downloading the files from their website.

Continuing with the 3D animation, in order to achieve the best performance for the tools is necessary to know about the computer architecture for 3D modeling and processing. In the actual market the computers are lifting their capabilities. Fabricants produces machines that are allowed to manipulate a huge amount of information in a minimal time, but depending of the architecture of the computer the performance can be limited by a bottleneck in a stage of the graphic rendering pipeline. The pipeline is divided in three stages: The Application stage where the data is read and where the models are updated. The Geometry stage is where the geometry is transformed and the illumination is calculated. And the Rasterizer stage where the 2D image is produced.

Finally the characters must be located in a virtual environment. In order to create a scene is necessary to know about how to create a 3D character (Geometric modeling), how to animate it (Cinematic modeling) and how to guide their actions in the virtual environment (Behavioral modeling). Each one of these kind of modeling provides different techniques and concepts. For example, the geometric modeling can be assumed from the polygons approach or from the splines approach; the cinematic modeling can be implemented by key framing or by importing external information (like motion capture); and the behavioral modeling places the character in a context and implements different algorithms to rule its actions.

In order to combine the two big arguments (Learning Theories and 3D animation) treated in a real-life context the project is developed inside a Visual Effects company in Milan: EDI Effetti Digitali Italiani. This company opened the doors to develop AI-based tools for 3D animation that are applied immediately. The software used to the creation is Houdini, a sidefx’s software that serves to create, animate, composite, etc. 3D environments and objects. This software has a complete user interface where several animations can be created. Houdini provides also a scripting interface using two different languages. Its own language Hscript allows the communication between nodes by matching parameters and attributes to generate special functionalities in the animations.

The scope of this project is to construct additional tools for Houdini to generate crowd simulations in static and dynamic situations and environments, taking actual and important techniques of 3D animation such as motion capture, rigging, python programming, UV texturing, level of detail management and behavioral modeling and combining it with the learning theories, in special the AI engineering approach.

Three tools are developed. The first one takes the manipulation and management of the motion capture library. For the dynamic case it is necessary to develop an artificial intelligence based algorithm, placing an agent into a virtual environment and making him read different variables and information from it.

Houdini provides a powerful tool that allows packaging an entire animation inside a reusable node and promoting parameters from the inner SOPs to manipulate the scene and change the outcome. After complete the entire design of the tools, an optimization for future applications is needed. Then it is important to create a reusable nature in the tools, in order to save time and money for future applications.
Finally are exposed the possible applications showing how is applied the static tool in the TIM falling star TV commercial. Also is shown an evaluation of the performance of the tools, analyzing the time that takes the machine calculating one frame varying the number of agents and the number of polygons.

**TEORETHICAL FRAMEWORK**

2 Learning and Knowledge

In psychology and education, learning is commonly defined as a process that brings together cognitive, emotional, and environmental influences and experiences for acquiring, enhancing, or making changes in one's knowledge, skills, values, and world views. Learning as a process focuses on what happens when the learning takes place. In this particular case, we must to center our attention in how the human being learns to then create a valid model to satisfy our propose: create a big crowd of people that can interact in a 3D environment. For this reason is important to know about the theories of how the knowledge is acquired by our brains and how reproduce it in a simple and light model.

2.1 The Cognitive Approach

2.1.1 Levels of analysis

A central tenet of cognitive science is that a complete understanding of the mind/brain cannot be attained by studying only a single level. Studying a particular phenomenon from multiple levels creates a better understanding of the processes that occur in the brain to give rise to a particular behavior. Marr gave a famous description of three levels of analysis:

1. *Computational theory*, specifying the goals of the computation;
2. *Representation and algorithm*, giving a representation of the input and output and the algorithm which transforms one into the other; and
3. *Hardware implementation*, how algorithm and representation may be physically realized.

2.1.2 Cognitive Approach on Artificial Intelligence

Artificial intelligence (AI) involves the study of cognitive phenomena in machines. One of the practical goals of AI is to implement aspects of human intelligence in computers. Computers are also widely used as a tool with which to study cognitive phenomena. Computational modeling uses simulations to study how human intelligence may be structured.

2.1.3 Cognitive Architectures

It proposes computational processes that act like certain cognitive systems, most often, like a person or acts intelligent under some definition. The term architecture implies an approach that attempts to model not only behavior, nut also structural properties of the modeled system.

In traditional AI, intelligence is often programmed from above: the programmer is the creator, and makes something and imbues it with its intelligence, though many traditional AI systems were also designed to learn. Biologically inspired computing, on the other hand, takes sometimes a more bottom-up, decentralized approach; bio-inspired techniques often involve the method of specifying a set of simple generic rules or a set of simple nodes, from the interaction of which emerges the overall behavior. It is hoped to build up complexity until the end result is something markedly complex. However, it is also arguable that systems designed top-down on the basis of observations of what humans and other animals can do rather than on observations of brain mechanisms, are also biologicaly inspired, though in a different way.¹

2.1.4 Intelligent Agents

In artificial intelligence, an intelligent agent (IA) is an autonomous entity which observes and acts upon an environment and directs its activity towards achieving goals. Intelligent agents may also learn or use knowledge to achieve their goals. They may be very simple or very complex: a reflex machine such as a thermostat is an intelligent agent, as is a human being, as is a community of human beings working together towards a goal.

Intelligent agents are often described schematically as an abstract functional system similar to a computer program. For this reason, intelligent agents are sometimes called abstract intelligent agents (AIA) to distinguish them from their real world implementations as computer systems, biological systems, or organizations. Some definitions of intelligent agents emphasize their autonomy, and so prefer the term autonomous intelligent agents. Still others considered goal-directed behavior as the essence of intelligence and so prefer a term borrowed from economics, "rational agent".

Intelligent agents in artificial intelligence are closely related to agents in economics, and versions of the intelligent agent paradigm are studied in cognitive science, ethics, the philosophy of practical reason, as well as in many interdisciplinary socio-cognitive modeling and computer social simulations.

According to the definition proposed by Russell and Norving (2003) the agents can be distinguish within five classes based on their degree of perceived intelligence and capability.

- **Simple Reflex Agents** act only on the basis of the current percept. The agent function is based on the condition-action rule: if condition then action.
- **Model-Based Reflex Agents** can handle partially observable environments. Its current state is stored inside the agent maintaining some kind of structure which describes the part of the world which cannot be seen.
- **Goal-Based Agents** are model-based agents which store information regarding situations that are desirable. This allows the agent a way to choose among multiple possibilities, selecting the one which reaches a goal state
- **Utility-Based Agents** only distinguish between goal states and non-goal states. It is possible to define a measure of how desirable a particular state is. This measure can be obtained through the use of a utility function which maps a state to a measure of the utility of the state.
- **Learning Agents**: Learning has an advantage that it allows the agents to initially operate in unknown environments and to become more competent than its initial knowledge alone might allow.²

Some other definitions have been done to classify the agents. Depending of the function of the agent can be classified as decision, input, processing, spatial, world, believable, physical or temporal agent.

### 2.2 Artificial Intelligence

#### 2.2.1 What is Artificial Intelligence?

Artificial intelligence is about making computers able to perform the thinking tasks that humans and animals are capable of. Some of the tasks that formerly were considered as AI (like computers programmed to have superhuman abilities in arithmetic, sorting and searching problem solving or to play games better than human beings) have been slipped out of the domain of AI, because those problems have been solved using more and more comprehensive ways. Nevertheless for some applications those algorithms aren’t good enough yet (for example, speak, recognize faces, etc).

#### 2.2.2 Games and Animation AI

The video games industry has developed a complex AI system, in parallel the animation industry has grown implemented some of this algorithms to manage an animation. Because both industries share most of the basis for implementation, during the following sections the specific AI techniques for this field will be explained.

Actually can be found a massive diversity of AI in which are addressed three basic needs: The ability to move characters, the ability to make decisions about where to move and the ability to think tactically or strategically.

#### 2.2.3 The kind of AI

The AI that is used for games and animations is divided into three different types and are more or less used in an equal way. These three types are hacking (ad hoc solutions and neat effects), heuristics (rules of thumb that only work in most, but not all, cases) and algorithms.

---

**Hacks:** The psychological correlate is behaviorism. It takes the behaviorism approach that was explained in the previous section.

**Heuristics:** Refers to experience-based techniques for problem solving, learning, and discovery.

**Algorithms:** Building algorithms to support interesting character behavior. General bits of AI, such as movement, decision making, and tactical thinking all benefit from tried and tested methods that can be endlessly reused.

### 2.2.4 Movement

One of the most fundamental requirements of AI is to move characters around. Even the earliest AI-controlled characters had movement algorithms that weren’t far removed from the games and animations on the shelf today.

#### 2.2.4.1 The basics of movement algorithms

Each character has a current position and possibly additional physical properties that control its movement. A movement algorithm is designed to use these properties to work out where the character should be next.

All movement algorithms have this same basic form. They take geometric data about their own state and the state of the world, and they come up with a geometric output representing the movement they would like to make. In all movement algorithms is needed an input, for example for obstacles avoidance it is necessary to know the coordinates of the obstacle.

#### 2.2.4.2 Two dimensional movements

Many animations have AI that works in two dimensions. Although animations rarely are drawn in two dimensions any more, their characters are usually under the influence of gravity, sticking them to the floor and constraining their movement to two dimensions.

A lot of movement AI can be achieved in just two dimensions, and most of the classic algorithms are only defined for this case. Before looking at the algorithms themselves, we need to quickly cover the data needed to handle two-dimensional (2D) math and movement.

Although a character usually consists of a three-dimensional (3D) model that occupies some space in the game world, many movement algorithms assume that the character can be treated as a single point. Collision detection, obstacle avoidance, and some other algorithms use the size of the character to influence their results, but movement itself assumes the character is at a single point. Collision detection and other forces can be applied to anywhere on the object, but the algorithm that determines the movement of the object converts them so it can deal only with the center of mass or just a point in space.

#### 2.2.4.3 Statics

Characters in two dimensions have two linear coordinates representing the position of the object. These coordinates are relative to two world axes that lie perpendicular to the direction of gravity and perpendicular to each other. This set of reference axes is termed the orthonormal basis of the 2D space.

In most animations the geometry is typically stored and rendered in three dimensions. The geometry of the model has a 3D orthonormal basis containing three axes: normally called x, y, and z. It is most common for the y-axis to be in the opposite direction of gravity (i.e., “up”) and for the x and z axes to lie in the plane of the ground. In addition to the two linear coordinates, an object facing in any direction has one orientation value (angle from a reference axis). Algorithms or equations that manipulate this data are called static because the data do not contain any information about the movement of a character.
2.2.4.4 Kinematics
So far each character has had two associated pieces of information: its position and its orientation. We can create movement algorithms to calculate a target velocity based on position and orientation alone, allowing the output velocity to change instantly.

A consequence of Newton’s laws of motion is that velocities cannot change instantly in the real world. If a character is moving in one direction and then instantly changes direction or speed, it will look odd. To make smooth motion or to cope with characters that can’t accelerate very quickly, we need either to use some kind of smoothing algorithm or to take account of the current velocity and use accelerations to change it. To support this, the character keeps track of its current velocity as well as position. Algorithms can then operate to change the velocity slightly at each time frame, giving a smooth motion.

2.2.5 Steering Behaviors
Steering behaviors extend the movement algorithms described above by adding velocity and rotation. They take as input the kinematics of the character that is moving and a limited amount of target information. The target information depends on the application. For chasing or evading behaviors, the target is often another moving character. Obstacle avoidance behaviors take a representation of the collision geometry of the world. It is also possible to specify a path as the target for a path following behavior.

The set of inputs to a steering behavior isn’t always available in an AI-friendly format. Collision avoidance behaviors, in particular, need to have access to the collision information in the level.

2.2.5.1 Path following
Path following is a steering behavior that takes a whole path as a target. A character with path following behavior should move along the path in one direction. Path following, as it is usually implemented, is a delegated behavior. It calculates the position of a target based on the current character location and the shape of the path. It then hands its target off to seek.

The target position is calculated in two stages. First, the current character position is mapped to the nearest point along the path. This may be a complex process, especially if the path is curved or made up of many line segments. Second, a target is selected which is further along the path than the mapped point by a fixed distance. To change the direction of motion along the path, we can change the sign of this distance.

2.2.5.2 Separation and Collision Avoidance
The separation behavior is common in crowd simulations, where a number of characters are all heading in roughly the same direction. It acts to keep the characters from getting too close and being crowded.

The collision avoidance behavior, below, should be used in this case. Most of the time, the separation behavior has a zero output; it doesn’t recommend any movement at all. If the behavior detects another character closer than some threshold, it acts to move away from the character. Unlike the basic evade behavior, however, the strength of the movement is related to the distance from the target. The separation strength can decrease. Separation is sometimes called the “repulsion steering” behavior, because it acts in the same way as a physical repulsive force (an inverse square law force such as magnetic repulsion). Where there are multiple characters within the avoidance threshold, the steering is calculated for each in turn and summed.

2.2.5.3 Obstacle and wall avoidance
The collision avoidance behavior assumes that targets are spherical. It is interested in avoiding getting too close to the center point of the target. This can also be applied to any obstacle in the game that is easily represented by a bounding sphere. Crates, barrels, and small objects can be avoided simply this way. More complex obstacles cannot be easily represented in this way. The bounding sphere of a large
object, such as a staircase, can fill a room. We certainly don’t want characters sticking to the outside of the room just to avoid a staircase in the corner.

The obstacle and wall avoidance behavior uses a different approach to avoiding collisions. The moving character casts one or more rays out in the direction of its motion. If these rays collide with an obstacle, then a target is created that will avoid the collision, and the character does a basic seek on this target. Typically, the rays are not infinite. They extend a short distance ahead of the character.

### 2.2.6 Decision Making

Decision making refers to the ability of a character to decide what to do. Carrying out that decision (movement, animation, and the like) is taken for granted. In reality, decision making is typically a small part of the effort needed to build great game AI. Most games use very simple decision making systems: state machines and decision trees. Rule-based systems are rarer, but important.

#### 2.2.6.1 Decision trees

Decision trees are the simplest decision making technique that we’ll look at, although extensions to the basic algorithm can make them quite sophisticated. They have the advantage of being very modular and easy to create.

A decision tree is made up of connected decision points. The tree has a starting decision, its root. For each decision, starting from the root, one of a set of ongoing options is chosen. Each choice is made based on the character’s knowledge. The algorithm continues along the tree, making choices at each decision node until the decision process has no more decisions to consider. At each leaf of the tree an action is attached. When the decision algorithm arrives at an action, that action is carried out immediately.

#### 2.2.6.2 State Machines

State machines take account of both the world around the characters (like decision trees) and their internal makeup (their state).

**Basic State:** In a state machine each character occupies one state. Normally, actions or behaviors are associated with each state. So, as long as the character remains in that state, it will continue carrying out the same action. States are connected together by transitions. Each transition leads from one state to another, the target state, and each has a set of associated conditions.

**Finite State:** A general system that supports arbitrary state machines with any kind of transition condition. The state machine will conform to the structure given above and will occupy only one state at a time.

**Mealy machine:** In the theory of computation, a Mealy machine is a finite-state machine whose output values are determined both by its current state and by the values of its inputs. The state diagram for a Mealy machine associates an output value with each transition edge.

**Moore machine:** In the theory of computation, a Moore machine is a finite-state machine whose output values are determined solely by its current state. The state diagram for a Moore machine associates an output value with each state.

#### 2.2.6.3 Fuzzy logic

This approach is very used in the entertainment industry. Software like Massive uses this concept to develop the agent’s brains. Conditions and decisions have been true or false, and the previous methods haven’t questioned the dividing line. Fuzzy logic is a set of mathematical techniques designed to cope with gray areas. Fuzzy logic allows blurring the line between two different actions, giving a whole spectrum of confidence levels. It doesn’t take into account to probabilistic methods, but permits to mix different behaviors without discard one.
The fuzzy logic can be used in any system where we’d normally have traditional logic AND, NOT, and OR. It can be used to determine if transitions in a state machine should fire. It can be used also in the rules of the rule-based system discussed. The algorithm doesn’t have a name. Developers often simply refer to it as “fuzzy logic.” It is taken from a sub-field of fuzzy logic called fuzzy control and is typically used to build industrial controllers that take action based on a set of inputs.

**Problem:** In many problems a set of different actions can be carried out, but it isn’t always clear which one is best. Often, the extremes are very easy to call, but there are gray areas in the middle. It is particularly difficult to design a solution when the set of actions is not on/off but can be applied with some degree.

![Exclusive mapping to states for fuzzy decision making](image)

The decision maker has any number of crisp inputs. These may be numerical, enumerated, or Boolean values. Each input is mapped into fuzzy states using membership functions as described earlier. Some implementations require that an input be separated into two or more fuzzy states so that the sum of their degrees of membership is 1. In other words, the set of states represents all possible states for that input.

### 2.2.6.4 Rule based systems

Rule-based systems were at the vanguard of AI research through the 1970s and early 1980s. Many of the most famous AI programs were built with them, and in their “expert system” incarnation, they are the best known AI technique. They have been used off on animations for at least 15 years, despite having a reputation for being inefficient and difficult to implement. They remain a fairly uncommon approach, partly because similar behaviors can almost always be achieved in a simpler way using decision trees or state machines.

Rule-based systems have a common structure consisting of two parts: a database containing knowledge available to the AI and a set of if–then rules. Rules can examine the database to determine if their “if” condition is met. Rules that have their conditions met are said to trigger. A triggered rule may be selected to fire, whereupon its “then” component is executed.

### 2.2.7 Artificial neural networks

Artificial neural networks (ANNs, or just neural networks for short) were at the vanguard of the new “biologically inspired” computing techniques of the 1970s. They are a widely used technique suitable for a good range of applications.

Neural networks consist of a large number of relatively simple nodes, each running the same algorithm. These nodes are the artificial neurons, originally intended to simulate the operation of a single brain cell. Each neuron communicates with a subset of the other artificial neurons in the network. They are connected in patterns characteristic of the neural network type. This pattern is the neural network’s architecture or topology.

#### 2.2.7.1 Feedforward and Recurrence

In many types of neural networks, some connections are specifically inputs and the others are outputs. The multi-layer perception takes inputs from all the nodes in the preceding layer and sends its single output value to all the nodes in the next layer. It is known as a feedforward network for this reason. The leftmost layer (called the input layer) is provided input by the programmer, and the output from the rightmost layer (called the output layer) is the output finally used to do something useful. Feedforward networks can have loops: connections that lead from a later layer back to earlier layers. This architecture is known as a recurrent network. Recurrent networks can have very complex and unstable behavior and are typically much more difficult to control. Other neural networks have no specific input and output. Each connection is both input and output at the same time.
2.2.7.2 Algorithm
The algorithm controls how a neuron should generate its state based on its inputs. In a multilayer perceptron network, the state is passed as an output to the next layer. In networks without specific inputs and outputs, the algorithm generates a state based on the states of connected neurons.

Each input has an associated weight. The input values are multiplied by the corresponding weight. An additional bias weight is added. The final sum is then passed through a threshold function. If the sum is less than zero, then the neuron will be off (have a value of zero); otherwise, it will be on (have a value of one). The threshold function can be smoother in order to achieve different states in the neurons.

2.2.7.3 Problem
Given a group of input values, the system must be in grade to classify the input in a different number of groups. As the situation becomes more complex, it can become difficult to create rules for decision trees or to develop a fuzzy state machine.

3 3D Animation

3.1 Motion Capture
Motion capture, motion tracking, or mocap are terms used to describe the process of recording movement and translating that movement onto a digital model. It is used in military, entertainment, sports, medical applications and for validation of computer vision and robotics. In filmmaking it refers to recording actions of human actors, and using that information to animate digital character models in 2D or 3D computer animation. When it includes face, fingers and captures subtle expressions, it is often referred to as performance capture.

The capture of the movement of a human actor to develop a model can be using the entire body, just the head or sometimes body parts like arms or legs. The main reason to do motion capture for 3D animation is that an animation done manually by key framing won’t be able to produce a realistic outcome, while motion capture ensures that the animation will be a copy of real movements.

Motion capture devices allow the recording of live motions by tracking a number of key points in space over time, which are translated into a 3 dimensional digital representation. The captured subject can be anything that exists in the real world, with the key points positioned on the object such that they best represent the orientations of the moving parts of the object, for example the joints or pivot points. In order to accurately triangulate marker positions at least 4 cameras are used, however generally no more than 32 are used.

The applications which motion capture is destined are worldwide, we can see motion captures in movies, in video games, in television, in some intelligent secure systems, in medical applications and also in sportive performance analysis.

3.1.1 Terminology and Notation
In order to understand and manipulate the information of a motion capture data file, it is necessary to define new concepts and keywords that will be useful to describe a motion.

**Skeleton:** The whole character for which the motion represents; the entire hierarchy of parts in which the character is divided.

**Bone:** The basic entity of a skeleton. The analogy with the bone structure of the animals is because each bone will represent the conformation of the body part. Each bone represents the smallest segment within the motion that is subject to individual translation and orientation changes during the animation. A skeleton is comprised of a number of bones, where each bone can be associated with a vertex mesh to represent a specific part of the character, for example the femur or humerus.

**Channel or Degree of Freedom (DOF):** Each bone within a skeleton can be subject to position, orientation and scale changes over the course of the animation, where each parameter
is referred to as a channel (or DOF). The changes in the channel data over time give rise to the animation.

**Frame:** Every animation is comprised of a number of frames where for each frame the channel data for each bone is defined. Motion capture data can be captured as high as 240 frames per second; however in many applications a rate of 30 or 60 frames per second tends to be the norm.

The model of transforming the bones (to set the position and orientation information correctly) is the arithmetic matrix operations. In this model is used a 4x4 matrix that is called the transform matrix (M), in this matrix is stored the information of the changes that will affect to the vertex. All the information of the vertexes of the bone is stored in floating point vectors (v) where the transform matrix is applied. After this operation the result is a new vector (v') with the transformation of the vertexes.

\[ v' = vM \]

Another convention that is very important when constructing a rotation matrix from its 3 separate Euler angles since matrix multiplication is not commutative is the rotation convention. The composite right to left rotation matrix, R, based on the separate rotation matrices about each axis, Rx, Ry, and Rz, where the composition order is “XYZ”. Since matrix multiplication is associative, brackets will be omitted from such equations.

\[ Rv = RxRyRzv \]

The motion of an individual bone consists of translation, rotation and scale components (depending on the channels defined for the bone), which can be merged together to give an overall transform using homogeneous coordinates. Unless otherwise stated, the combination order of these different transforms to give the full transform.

\[ M = TRS \]

Where S, R and T are the scale, rotation and translation matrices respectfully.

In most motion capture file formats, the data is presented in a hierarchical manner and this formula only gives the local transformation of a bone. The local transformation of a bone describes its orientation within in its local coordinate system, which in turn is subject to its parent’s local orientations. To obtain a global matrix transform for a given bone, the local transform needs to be pre-multiplied by its parent’s global transform, which itself is derived my multiplying its local transform with its parent’s global transform and so on.

\[ M_{\text{global}}^n = \prod_{i=0}^{n} M_{\text{local}} \]

### 3.1.2 Methods and Systems

#### 3.1.2.1 Optical Systems

Optical systems utilize data captured from image sensors to triangulate the 3D position of a subject between one or more cameras calibrated to provide overlapping projections. These systems produce data with 3 degrees of freedom for each marker, and rotational information must be inferred from the relative orientation of three or more markers; for instance shoulder, elbow and wrist markers providing the angle of the elbow.

**Passive Markers:** Use markers coated with a retroreflective material to reflect light back that is generated near the cameras lens. The camera's threshold can be adjusted so only the bright reflective markers will be sampled, ignoring skin and fabric.

---

3 Meredith M., Maddock S., “Motion Capture File Formats Explained”, p. 1-9,
**Active Markers:** These systems triangulate positions by illuminating one LED at a time very quickly or multiple LEDs with software to identify them by their relative positions, somewhat akin to celestial navigation. Rather than reflecting light back that is generated externally, the markers themselves are powered to emit their own light. Since Inverse Square law provides $1/4$ the power at 2 times the distance, this can increase the distances and volume for capture.

**Markerless:** Markerless systems do not require subjects to wear special equipment for tracking. Special computer algorithms are designed to allow the system to analyze multiple streams of optical input and identify human forms, breaking them down into constituent parts for tracking. Products currently under development include Microsoft's Kinect system for PC and console systems.

### 3.1.2.2 Mechanical Systems
Mechanical motion capture systems directly track body joint angles and are often referred to as exoskeleton motion capture systems, due to the way the sensors are attached to the body. Performers attach the skeletal-like structure to their body and as they move so do the articulated mechanical parts, measuring the performer’s relative motion. Mechanical motion capture systems are real-time, relatively low-cost, free-of-occlusion, and wireless systems that have unlimited capture volume. Typically, they are rigid structures of jointed, straight metal or plastic rods linked together with potentiometers that articulate at the joints of the body.

### 3.1.2.3 Magnetic Systems
Magnetic systems calculate position and orientation by the relative magnetic flux of three orthogonal coils on both the transmitter and each receiver. The relative intensity of the voltage or current of the three coils allows these systems to calculate both range and orientation by meticulously mapping the tracking volume. The sensor output is 6DOF, which provides useful results obtained with two-thirds the number of markers required in optical systems; one on upper arm and one on lower arm for elbow position and angle. The markers are not occluded by nonmetallic objects but are susceptible to magnetic and electrical interference from metal objects in the environment, like rebar (steel reinforcing bars in concrete) or wiring, which affect the magnetic field, and electrical sources such as monitors, lights, cables and computers. The sensor response is nonlinear, especially toward edges of the capture area. The wiring from the sensors tends to preclude extreme performance movements. The capture volumes for magnetic systems are dramatically smaller than they are for optical systems. With the magnetic systems, there is a distinction between “AC” and “DC” systems: one uses square pulses, the other uses sine wave pulse.\(^4\)

### 3.1.2.4 Inertial Systems
Inertial Motion Capture technology is based on miniature inertial sensors, biomechanical models and sensor fusion algorithms. The motion data of the inertial sensors (inertial guidance system) is often transmitted wirelessly to a computer, where the motion is recorded or viewed. Most inertial systems use gyroscopes to measure rotational rates. These rotations are translated to a skeleton in the software. Like the optical markers, the more gyros, the more natural the data. No external cameras, emitters or markers are needed for relative motions. Inertial mocap systems capture the full six degrees of freedom body motion of a human in real-time. Benefits of using inertial systems include: no solving, portability, and large capture areas. Disadvantages include lower positional accuracy and positional drift which can compound over time.

These systems are similar to the Wii controllers but are more sensitive and have greater resolution and update rates. They can accurately measure the direction to the ground to within a degree. The popularity of inertial systems is rising amongst independent game developers, mainly because of the quick and easy set up resulting in a fast pipeline.

### 3.1.3 Motion Capture Database – Carnegie Mellon University
The Carnegie Mellon University Graphics Lab has developed an important free motion capture database, that is used in the project because of its compatibility with Houdini’s tools. The process of the production of a motion capture is divided in several steps, from the motion capture lab equipment to the file format writing.

The motion capture lab in the basement of Wean contains 12 Vicon infrared MX-40 cameras (http://www.vicon.com/), each of which is capable of recording 120 Hz with images of 4 megapixel resolution. The cameras are placed around a rectangular area, of approximately 3m x 8m, in the center of the room. Only motions that take place in this rectangle can be captured. If motion of human hands is being captured, more detail is required and the cameras are moved closer to capture a smaller space with higher resolution.

To capture something, small grey markers are placed on it. Humans wear a black jumpsuit and have 41 markers taped on. The Vicon cameras see the markers in infra-red. The images that the various cameras pick up are triangulated to get 3D data.

This 3D data can be used in two ways:

**Marker positions:** The information given as the outcome of the motion capture process is the 3D position of each marker and it’s given in a .c3d file. This file is highly manipulable in sense of renaming the markers. The problem is that the information presented in this file doesn’t relate the information between markers, all are independent.

**Skeleton movement:** The outcome of the process will be the either a .vsk/.v file pair or .asf/.amc file pair. The first file describes the skeleton and its joints: their connections, lengths, degrees of freedom and mathematical transformations. The other file is the description of the movement. If there are more than one motion capotes for a single agent, the package will contain several files of the second class (.v or .amc). Every capture will deliver a skeleton, no matter if there is just one bone.

After the capture, the rest of the process is to manipulate the .v or .amc file within the software that is provided by Vicon that is called “ViconIQ”.

Vicon must be told what skeleton to use, in the form of a .vst, a Vicon Skeleton Template. These can be created in ViconIQ itself, under the modeling tab. The Vicon software comes with documentation on editing them. Visualized, they look like maya skeletons covered in porcupine needles. They specify the skeleton hierarchy, and what markers will be captured to help construct this skeleton. They give approximate bone lengths - the actual length, of course, will depend on the subject/object being captured.

The markers are carefully placed to get maximal information - consider that if you had a hinge joint, 2 or 3 markers would define it absolutely. Constraints between markers and joints are also specified, e.g. "the elbow belongs at the y-location of this marker", or "the wrist joint is halfway between these two markers". You get the idea. Constructing .vst's for complex objects requires careful thought and testing.

ViconIQ requires user interaction to start off the skeleton fitting. To process a capture, a segment of motion is loaded onscreen as a point cloud of markers. The user goes through and specifies the correspondence between these markers and the markers in the .vst, e.g. "this white dot is the clavicle marker". From this data ViconIQ can fit a skeleton and determine the skeleton's limb lengths. From here on out the labeling process is automatic. ViconIQ can load up each motion clip and automatically perform a "Kinematic Fit" of the skeleton to the markers. During this time the software uses its knowledge of the skeleton to correct captured marker aberrations. The user can also fix things up by editing the joint rotation/translation graphs directly.

While this work is going on each motion clip is stored in a .trial file. When the data is clean, it is time to export useful files. A .vsk of the skeleton is exported. Keep in mind that this .vsk is unique to each person, because each person has different limb lengths. Multiple .v's are exported, one for each motion clip the person performed. Using BodyBuilder, these can be turned into asf/amc’s.

http://mocap.cs.cmu.edu/info.php
3.1.4 File Formats

In the real world, many motion capture techniques are used worldwide. But there is a problem that involves the standardization of the motion capture file formats. Because this tool is used recently there are many companies that have developed their own motion capture system, coding the data in their own way and making it incompatible with the others file formats. However the ASCII nature of many of the formats makes it reasonably easy to decode and understand by simple inspection of the data.

The file formats that are used in the database beyond the ViconIQ files (.vsk/.v) are the .asf/.amc and with the pertinent conversion, the .bvh files. The conversion of the original files is necessary to have an interoperability of the data within the 3D animation programs. The following section makes an overview of this file formats explains how to parse them.

3.1.4.1 Acclaim ASF/AMC

Acclaim is a game company which has been doing research into motion capture for games for many years. They developed their own methods for creating skeleton motion from optical tracker data and subsequently devised a file format, actually two files, for storing the skeleton data. Later they put the format description in the public domain for anyone to use. Oxford Metrics, makers of the Vicon motion capture system, elected to use the Acclaim format as the output format of their software.

The Acclaim format is made up of two files, a skeleton file and a motion file. This was done knowing that most of the time a single skeleton works for many different motions and rather than storing the same skeleton in each of the motion files it should be stored just once in another file. The skeleton file is the ASF file (Acclaim Skeleton File). The motion file is the AMC file (Acclaim Motion Capture data).

In the ASF file a base pose is defined for the skeleton that is the starting point for the motion data. Each segment has information regarding the way the segment is to be drawn as well as information that can be used for physical dynamics programs, inverse kinematic programs or skinning programs. One of the peculiar features of the ASF file is the requirement that there be no gaps in the skeleton. No child can have a non-zero offset from the end of the parent segment. This has the effect of creating more skeletal segments than are usually found in other file formats. A limitation of the ASF definition is that only one root exists in the scene, this doesn't prevent a file from cleverly containing two skeletons attached to the root but it does make such a construction clumsy.

The AMC file contains the motion data for a skeleton defined by an ASF file. The motion data is given a sample at a time. Each sample consists of a number of lines, a segment per line, containing the data. The start of a sample is denoted by the sample number alone on a line. For each segment the segment name appears followed by the numbers in the order specified by the dof keyword in the ASF file.

For each segment it is useful to precalculate some of the transformation matrices that will be used to construct a global transform for a segment. First create a matrix \( C \) from the axis using the axis order to determine the order the rotation values are composed. In the ASF file the order is given left to right so that an order of "XYZ" is:

\[
\nu M = \nu XYZ
\]

Do this same calculation for the root but use the orientation value with the axis order for the root. After calculating \( C \) take the inverse of \( C \), call it \( C_{inv} \), and save it.

Next create a matrix \( B \) and from the translation offset from the segments parent. The translation offset is the direction and length of the parent segment. For the root use the position value. This concludes the precalculation step.

When you are constructing the transformation matrix of motion for a segment, first create a matrix, \( M_i \), of the motion data. When creating \( M \) construct a separate matrix for each dof specification and multiply them together left to right. Compose the local transform, \( L \), by multiplying \( M \) on the left by \( C_{inv} \) and on the right by \( C \) then \( B \):

\[
L = C_{inv}MCB
\]
Like with other formats create the full transform by traversing the hierarchy and multiplying on the right by each parent in the skeleton.\(^6\)

### 3.1.4.2 BioVision: BVH (BioVision Hierarchical data)

The file .bvh takes the problem of describing the skeleton with other paradigm. This format only delivers one file in which the information of the skeleton and the motion are both included. It specifies the hierarchy and the parameters order that must be followed until the end of file, to describe correctly the translation and rotation of every single bone in the hierarchy.

The BVH format succeeded BioVision’s BVA data format with the noticeable addition of a hierarchical data structure representing the bones of the skeleton. The BVH file consists of two parts where the first section details the hierarchy and initial pose of the skeleton and the second section describes the channel data for each frame, thus the motion section.

The hierarchical section of the file starts with the keyword HIERARCHY, which is followed on the next line by the keyword ROOT and the name of the bone that is the root of the skeletal hierarchy. The ROOT keyword indicates the start of a new skeletal hierarchical structure and although the BVH file is capable of containing many skeletons, it is usual to have only a single skeleton defined per file.

The remaining structure of the skeleton is defined in a recursive nature where each bone’s definition, including any children, is encapsulated in curly braces, which is delimited on the previous line with the keyword JOINT (or ROOT in the case of the root bone) followed by the name of the bone. With the introduction of a left curly brace it is good practice to indent the bone’s content (with a tab) and align the closing curly brace with the corresponding opening one. The bone names identified by the prefix JOINT or ROOT are not referenced again in the file and hence redundant, however some parses require a bone name in order to correctly parse the file. Furthermore, although the hierarchical indentation is not absolutely necessary, it does assist in making the file more readable for humans.

Within the definition of each bone, the first line, delimited by the keyword OFFSET, details the translation of the origin of the bone with respect to its parent’s origin (or globally in the case of the root bone) along the x, y and z-axis respectively. The offset serves a further purpose of implicitly defining the length and direction of the parent’s bone, however the problem with this is in defining the length and direction of a bone that has multiple children. Normally a good choice for determining the bone length in this situation is to use the first child offset definition to infer the parental bone information and treat the offset data for other child nodes simply as offset values.

The second line of a bone’s definition is prefixed with the keyword CHANNELS which defines the DOFs for the current bone. The importance of the order that the channels are presented is two-fold. First, the order that each channel is seen in the hierarchy section of the file exactly matches the order of the data in the motion section of the file. For example, the motion section of the file contains information for the channels of the root bone in the order defined in the hierarchy, followed by the channel data for its first child, followed by the channel data for that child and so on through the hierarchy. The second point to note with regards to the channel ordering is that the concatenation order of the Euler angles when creating the bone’s rotation matrix needs to follow the order depicted in the CHANNEL section. It is important to note this because the Euler order is specified for each bone; therefore it is possible to have different orders for different bones, which needs to be accounted for in order to get a correct looking animation.

After the OFFSET and CHANNEL lines, the next non-nested lines in the bone definition are used to define child items, starting with the keyword JOINT, however in the case of end-effectors, a special tag is used, “End Site”, which encapsulates an OFFSET triple that is used to infer the bone’s length and orientation.

Once the skeletal hierarchy is defined, the second section of a BVH file, which is denoted with the keyword MOTION, contains the number of frames in the animation, frame rate and the channel data. The line containing the number of frames starts with the keyword “Frames:” which is followed by a positive decimal integer that is the number of frames. The frame rate is on a line starting with “Frame Time:” which is followed by a positive float that represent the duration of a single frame. To convert this into a frames per second (FPS) format you simply need to divide 1 by the frame time. Once the number of frames and frame time has been defined, the rest of the file contains that channel data for each bone in the order they were seen in the hierarchy definition, where each line of float values represents an animation frame.

### 3.2 Computer Architecture for Virtual Reality

In this part we will take care about the structure of the hardware that is used for virtual reality in real time interactive applications. The main functions of the computer (from here VR engine) are to read the data that is sent from the input dispositive, develop the applications relative to them and the produce an outcome that will be interpreted in the output dispositive.

The VR Engine could be a single computer, a computer local networked or a remote computer network in which each computer has a specific function. For a VR simulation it is not possible to predict every single action done by the user, so to create the virtual world in real time we need to refresh the output in the time range (25/30 frames or 40/33ms), to refresh the feedback tactile quick and to have a real low delay of the system. For this reason the importance from the analysis of the rendering pipeline, that for virtual reality applications is not anymore just associated with graphics, to be associated also with the audio and tactile outcomes.

Now let’s see the rendering pipeline for graphics, analyzing the possible bottlenecks, how to optimize.

The graphic rendering is the process that generates a 2D image from a 3D representation. This process is effectuated inside a sequential 3 stage. The speed of the pipeline will be limited by the speed of the lowest stage in this sequence. The Application stage is the stage where the data is read and where the models are updated. The Geometry stage is where the geometry is transformed and the illumination is calculated. Finally in the Rasterizer stage the 2D image is produced.

To improve the specifications it is introduced the concept of parallelism inside the different stages. It consists in divide the entire work into little pieces that can be done independently by independent tasks. In the application stage the parallelism is defined by the number of computers that are used, while in the geometry and rasterizer stages, the parallelism is treated by the architecture of each dispositive that is used. I.e. For the Geometry stage it is created a FIFO buffer in the input that will send the information to the different Graphic Engines (GE) inside. For the Rasterizer stage an input and an output buffers are created as well, and the imaged is treated inside different rasterizer units.
3.2.1 Stages

3.2.1.1 Application Stage
It is 100% done in software level in the CPU. It reads the Data Base of the geometry and the input data that comes from the devices. Based in the user inputs will produce a different outcome by modifying the simulation view or changing the position or orientation of the virtual objects. Beyond it identifies the collisions and generates an appropriated feedback to the tactile output device.

To achieve an optimization from this stage, it’s necessary to reduce the complexity of the models that is measured in terms of number of polygons. That action implies a reduction of the information that will be processed in the pipeline.

Other actions that also can take place for optimize the application stage are: To change the precision of the float point data, minimize the number of divisions and it is strongly recommended to utilize a multi-processor or multi-core architecture in the CPU.

3.2.1.2 Geometry Stage
It is implemented in software and hardware. In this stage the transformations of the model like scale, rotation and translation are done, the illumination is calculated, the scene is projected and the texture maps are applied. Nowadays, the new Graphic Cards are designed to do this entire process in hardware using the implementation of the OpenGL and DirectX.

The most complex process on this stage is the calculation of the illumination. It calculates the superficies color based in the type and the number of the synthetic lights, the illumination model, the superficies proprieties (like reflection or alpha channel) and the special effects that have been put in the scene (like motion blur, smoke, rain, etc). As a result the Geometry Stage produces the shadowing effect that makes the scene much more realistic.

To optimize this process it is recommended to reduce the number of the lights in the virtual environment. Simplify the model of illumination also reduce the computational cost of the scene, but also reduce the realism of the scene. These models are often used to do some tests, because they show a very good approximation to the final scene, these models are implemented in all the 3D modeling software.
The way that the supericies are described is directly nested to the speed of rendering. For example, the hardware is done to process faster triangular primitives, if the supericies are described by triangular mesh, the rendering time will be shorter than the rendering time of the supericies that are described with polygonal primitives with a higher number of edges.

3.2.1.3 Rasterizer Stage

It is done 100% in hardware. It converts the 2D geometrical information of the model vertexes associated to the information of the colors that have been calculated in the Geometry Stage to color information for every single pixel needed to visualize the video. The z component is saved in a called z-buffer, and it serves to ensure that the objects that are visualized from the camera would be visualized in the right way.

This stage usually utilizes a double-buffer technique, where the scene rendering arrives to the back buffer and when the process is finished the final image is stored in the front buffer. This technique completely avoids the flickering of the image that is produced by the access at the same time for the read/write functions within the video memory. Every single buffer of the system is finally grouped in the frame buffer.

For avoid the problems presented by the aliasing, it’s necessary to implement an antialiasing system. In this system each pixel is divided in n sub-pixels, for each one of them is calculated a color value. The final color value of the pixel is the mixture of the colors calculated previously. Obviously the big is the number of samples, better is the antialiasing system.

3.2.2 Graphic Station Architecture

Nowadays PCs represent the most used device to execute graphic applications, the fast diffusion of videogames and simulations has made that the performance of the computers increases in an exponential way. The parts of the PC that are involved in the execution of graphic applications are the CPU (speed and parallelism) and the graphic cards (measured in polygons per second or FLOPS, Floating Point Operation per Second).

The processors everyday turn more and more powerful, so do the graphic cards. We can see that in 1994 we had the 486 Intel that worked at 66MHz, while in 2009 was launched the Intel Core i5 multi-core, and the graphic cards also have had an impressive increment in their performance, passing from 7000 G-Shaded Pol/sec in 1994 to 27Million G-Shaded Pol7sec in 2004. The reason is that the videogames have become more and more complex and realistic, and also work in real time, so the Graphic cards have to calculate complex algorithms of shading and visual effects.

This evolution has influenced the development of the buses to communicate between the different parts of the PC (CPU, I/O devices and memory). Actually for graphic applications the buses have arrived to transmit 32GB/s with the PCI Express bus.

Another part that represents a very important component for a PC based graphic station is the Graphics Accelerator. It is a dedicated hardware for the generation of the output images that is plugged to the PC through a dedicated bus. The Graphics Boards are composed mainly by:
Graphics Processing Unit (GPU): It’s a microprocessor dedicated to the graphic rendering (Rasterizer Stage), it is optimized for the management of the floating point operations. The rendering consists in generating an image 60 times per second.

Video BIOS: Contents the programs to manage the board and the communication with the I/O devices.

Video memory: Is the RAM used for the video buffer, z buffer and texture maps.

RAMDAC: RAM Digital to Analog Converter converts the digital signal to an analogical output signal.

3.3 Modeling
An important part for virtual reality applications is the 3D modeling of objects and characters. It consists in the creation of the objects that will populate the virtual environment, by generating a complete definition for their superficial, physic and cinematic characteristics and the definition of their behavior. To manage the creation of models, are used specific for virtual reality authoring tools (tools for describing contents) that can be programs or software libraries.

The cycle of 3D object modeling is composed by several steps that are not always necessary to be done in their totality. These steps are:

- Geometric Modeling: Population of the virtual world, material and texture definitions.
- Cinematic Modeling: The way the objects move.
- Physic Modeling: Physic properties like weight, inertia, etc.
- Interaction between I/O devices and objects.
- Behavior Modeling: Creation of intelligent agents.
- Model Management: Model optimization.

![Diagram of 3D object modeling](image)

**Figure 6 3D object modeling**

3.3.1 Geometric Modeling
It is composed by two big stages. The former one is the definition and description of the object superficies, using polygonal meshes and Splines. The latter one is the object aspect definition where the materials, the shadowing and the texture mapping are described.

3.3.2 Cinematic Modeling
The cinematic modeling describes the way the objects move and determines the position and the movement of the 3D objects in the scene, referenced to a fixed coordinate system called world coordinate system. The first aspect of the cinematic modeling regards to the hierarchies in the objects, all the movements of the parent will affect directly the movement of the children. Other important aspect is the way the scene is viewed and the management of the virtual camera.

To calculate the transformations in the virtual world the mathematical model that has been developed use a reference coordinate system defined from three orthonormal unit vectors (i, j, k) that must have this properties:

\[ |i|=|j|=|k|=1 \quad \& \quad i.j=i.k=j.k=0 \]
The homogeny transform matrices have dimension 4x4 and they put the relation between two different homogenous coordinate systems and their format is:

\[
T_{A\leftarrow B} = \begin{bmatrix}
R_{3\times3} & P_{3\times1} \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Where \( R_{3\times3} \) is the rotation sub-matrix and represents the direction of the B system respect the A system. And \( P_{3\times1} \) is the position vector and represents the B system origin respect the A system.

With this matrices approach, the model takes all the transformations in just one representation. Also with the properties of the matrix the manipulation of them is very easy, for example the matrices can be inverted easily.

The model of the object uses a reference system associated to the object itself usually located in the gravity center of the object. When an object is moving in the virtual space, its reference system is moving with it, so the position and direction of the vertices of the object in its reference system are kept without changes, only will be changes if the object is deformed.

Within the world reference system, position and direction of the object are defined by a transform matrix that put a relation between both coordinates system (world’s one and object’s one).

\[
T_{W\leftarrow L} = \begin{bmatrix}
l_1 & l_2 & l_3 & I_4 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Where \( I, j, k \) are 3x1 vectors that projects the vectors of the object reference system into the world reference system. When the object is moving the transform matrix depends of the time.

A hierarchy defines a group of objects that will move at the same time but their own components can be moved independently. There are two levels of virtual objects, the parent and the child. The movement of the parent is propagated to its children, but the movement of the children is not propagated to its parent. Also this hierarchy is described by the use of 4x4 matrices. This hierarchy can be graphically represented by a tree, where the nodes are the hierarchy components and the unions are the relation between them. In the top of the tree there are the global transformations and it determines the scene view, when it is modified, all the objects are modified.

### 3.3.3 Behavior modeling

The behavior modeling makes reference to how the objects act within the scene. Inside the simulation it is possible to define objects that its behavior won’t be controlled by the user, but will be autonomous. The Level of Autonomy (LOA) of the entire scene will depend of the LOA of every single agent, can be interactive objects, intelligent agents or agent’s groups. According to the Thalmann definition there are three types of LOA:

**Guided:** The behavior of the agent is defined from an external process. Here the agent needs to receive during the simulation a list of collision free positions and the information necessary to apply an action. For example, an automatic door that opens when someone is getting closer.

**Programmed:** The possible actions that the agent can take are predefined. The Agent is programmed to follow a path while avoiding collision with other agents and programmed obstacles and to manage where and how an action can occur. For example, a character that moves following a fixed way.

**Autonomous:** The agent collects the information from the environment and takes a decision about what to do. The agent is able to perceive information in the environment and decide a path to follow to reach the goal, using the environment perception or the memory. For example, a soccer player that moves in the ground in a videogame.

The **interactive objects** have a behavior that doesn’t depend from the user inputs. It will depend from a timer or sensors placed in the virtual environment. For example, when you need a door to be opened you just need to place a proximity sensor that will send a signal to open it. The **intelligent agents** are entities that have a behavior guided by some rules for the inputs of the world. Those inputs
are the perceptions, emotions and behavior and are produced from the interaction with the virtual environment. As the interactive objects, the intelligent agents “read” the virtual world by using sensors, but the action to take, will depend from predefined rules. The group of agents is a crowd that can be guided, programmed or autonomous.

In the case of **group of agents or crowd simulation**, usually we intend to have lots of virtual human agents avoiding dealing with individual behaviours. Contrary to the individual case, our goal here is to describe methods to provide intelligence focused in a common group entity that controls its individuals. It is called **groups-based application**, the crowd and group applications, where individual complexity is less required. In this case, the intelligence abstraction can be included in the groups providing more autonomy to the groups instead to the individuals.

Considering levels of autonomy (LOA), Thalmann, Kallmann and Musse have developed a model for classify the crowd behaviors. They have classified the crowd behavior in three kinds:

i) **Guided crowds**, which behaviours are defined explicitly by the users;

ii) **Programmed crowds**, which behaviours are programmed in a script language;

iii) **Autonomous crowds**, which behaviours are specified using rules or others complex methods.\(^7\)

Many studies to develop models for the selection of actions have been done in the last years. Models like the motivational model proposed by de Sevin and Thalmann in 2005 in which is selected the better behavior for the situation based on the motivations of the agent. It is based in a hierarchic classification system working in parallel to obtain goal-oriented behaviors for virtual humans. This model contains four levels per motivation: Internal variables, Motivation, Motivated Behavior and Action.\(^8\)

**SPECIFICATIONS**

### 4 Crowds Simulation Project

#### 4.1 Overview

With the development of new technologies for animation and visual effects, and the growth of the capacity of process of the computers, new tools must be created to impress the market, which has lift its expectancies a lot; with innovative and interesting solutions. During the last years, the level of the productions and the quality of the effects has increased notoriously, showing an important evolution and highlighting the necessity of making the things in a better and faster way. The cinema, advertising and television markets actually require a fast and effective production of new proposals, so the creation of new tools and complements to the actual systems seems to be a big and open field to follow and fill.

One of the effects that are more intriguing is the multiplication of agents within a scene. In most cases it’s necessary to fill a space, a location or a place with people that really don’t make a lot for the story but are really necessary to give a sense of realism of a scene. Once, this effect was done with the use of extras, real people that was paid to be on the background of the scene without any argument or dialog. It was a really hard work because a human error can occur in any moment and the entire scene had to be redone. Nowadays the incursion into the market of new techniques for human animation such as motion capture, mixed with the use of geometries with a really high number of polygons that gives the model a more realistic sensation allows to replace real people with 3D copies. This action has several implications into the production pipeline and the budget of the project. The time of rent of cameras, lights and other stuff used during the recording is reduced, but the time of post-production raises. Nevertheless it implies a reduction on the cost of the rent of equipment that might be higher than the increase of the postproduction cost.

---

\(^7\) Musse S., Kallmann M., Thalmann D. “Level of Autonomy for Virtual Human Agents”. 1999

\(^8\) De Sevin E., Thalmann D. “An Affective Model of Action Selection for Virtual Humans”. 2005
The project is developed inside an Italian visual effects company called EDI that produces the effects for cinema and advertising. The company uses the software Houdini to create the 3D animations and models. The main scope of this project is to generate additional tools for Houdini involving one of the most important weapons of the 3D animation field: the crowd replication and simulation.

The project represents the integration of the most important and useful tools and techniques of the cinematographic industry in order to create a fast, ready and reusable product. Within the techniques used that can be found there are: 1. the motion capture for the human animation simulation; 2. rigging for geometry deformation; 3. artificial intelligence and behavioral modeling for the integration of the agents between them and with the environment; 4. and the use of Houdini’s mantra rendering tool to generate the final product.

4.2 General Description

The project is divided in four sequential stages. Each one of these stages has been developed with duration of about a month.

The first stage is referred to the learning of the Houdini’s environment and its related tools. Within this stage several tutorials about the Houdini’s interface, nodes, techniques and programming were done in order to get experience with the application. Also tutorials about Hscript, global and standard variables, Python and communication between them were followed. Most of those tutorials were found on the sidefx website (sidefx are the developers of Houdini) and others were provided by EDI from their own library. Then is implemented the management of the motion capture files. As it is described on the part one, there is used a free library created by the Carnegie Mellon University. This section also was guided by a tutorial bought by EDI from vfxtoolbox.com, and is very useful to learn about techniques of communication within different node types and the use of the programming languages inside Houdini. All the development of this section is described later on the Casting.otl section.

The second stage is the first practical application of the techniques integration. It represents the construction of an additional tool for Houdini to create simulations of static crowds. There is used the knowledge acquired in the two previous stages and all its explanation and development is contained on the CrowdReplication.otl section.

The third stage is the design and implementation of an artificial intelligence based algorithm for the creation of simulated dynamic crowds. Later on the CrowdReplicationDyn.otl section is deeply explained the procedure taken to achieve a working algorithm.

The fourth stage is the error correction and performance analysis section. After the construction of the new tools it is necessary to think about the errors and limitations that can be presented and try to generate an effective solution for those problems. So in this stage are implemented solutions to improve the performance of the tools and to correct some mistakes that affect the complexity of the calculus in the computer, achieving satisfactory results.

With this order of ideas the scope of this project is to construct additional tools for Houdini to generate crowd simulations in static and dynamic situations and environments, taking actual and important techniques of 3D animation such as motion capture, rigging, python programming, UV texturing, level of detail management and behavioral modeling. Also for the dynamic case it is necessary to develop an artificial intelligence based algorithm, placing an agent into a virtual environment and making him read different variables and information from it.

After complete the entire design of the tools an optimization for future applications is needed. Then it is important to create a reusable nature in the tools, in order to save time and money for future applications, by creating the possibility of store the time-consuming actions to avoid the repetition in the later projects. In parallel, during the development of the project it is expected to learn about a production pipeline for cinema and advertising within a professional and real context, making experience by applying the designs in different projects.
4.3 About EDI
EDI Effetti Digitali Italiani takes care about advise, supervision and production of digital visual effects.

EDI has born in Milan in February 2001. Initially its production was only focused on the advertising field, producing visual effects for major production companies and more established filmmakers. Since 2002, starts to involve the cinematographic market and in parallel develops a dedicated area to the post-production service.

Its main strength is photorealism. Since 2003, EDI moved to Via Giordano Bruno 15 in a 3 floor building in the heart of Chinatown where has about 25 stable employees. Since its founding EDI has brought a disruptive element in the field of visual effects in Italy, because of the new methodology in dealing with production and research of the highest quality in integrating the effects on the recording.

EDI also has always favored the research and development of new tools by working steadily on Linux, writing the necessary tools and maintaining a team of professionals capable to resolve the problems that come from the production side. The direction is established to look for interesting and innovative solutions in accordance with the specifications of the director or the customer, putting their experience and passion as useful help and not only the execution.

Concluding EDI placed on the market in order to take a project from its birth with the implementation of storyboards and animations, doing also the supervision on the set, film editing offline and online until the realization of the broadcast material.

With this context the following three chapters will deepen into the creation, design and evaluation of the tools in Houdini and their possible applications in the real life.

DEVELOPMENT AND ANALYSISIS

5 Casting.otl

5.1 Introduction
The theoretical base that is applied here is the real time management for a virtual environment. Also the main topic of this digital asset is the Motion Capture technique. In order to reach a realistic animation for human movements, the motion capture technique brings an important option. Nowadays this technique is used in all the multimedia fields because its simplicity to manage the files. Within the applications are included special effects for movies such as “Avatar” or “The Curious Case of Benjamin Button”, movements for video games such as “Call of Duty” or “Pro Evolution Soccer”. As it is referenced in the theoretical framework, the implementation of this technique has a high cost for the first time, but with the infrastructure constructed the cost of making one capture is the same of the cost of making as many captures as needed.

Also it serves as the introduction for the Houdini potential use. Many features are strongly used to acquire experience and knowledge to the future use in the creation of digital assets. Knowledge about Houdini’s nodes, Hscript, Python and Houdini’s interface is taken from tutorials as part of the formation into the complete utilization of the environment brought by Houdini.

Another argument that is highly treated during this digital asset is the management of the real time on the virtual environment. The main idea is to achieve the real time view of many animations at the same time, so the management of the resources has to be the main thing. In order to reduce the complexity of the calculus Houdini brings some alternatives to reach the same result using less resources, like copying the skeleton is less complex than create twenty different skeletons. This digital asset has not a render images purpose so the success of this digital asset is to have a fluid sequence of images on the View Pane.

The final argument is the creation of a package of nodes that are only manipulated from external parameters in order to change the outcome on the View Pane. This package is called digital asset and serves to share its functionalities with other computers for other situations.
5.2 General Description
Casting.otl is a Houdini 10 digital asset that reads and takes all the animations on a Motion Capture Library in order to show them in an ordered and easy way. This digital asset is created as a complement to the design of crowd replications digital assets.

The main objective of this digital asset is to choose the animations that will be part of our crowd replication system. It is not the animation of the crowd! It is a tool that will help us to select the animations. After the selection stage, it is necessary to make some changes to the animations.

With a simple selection of inputs you can manipulate this digital asset for a satisfactory visualization of the animations. (It works on Houdini Master and Houdini Escape due to it only uses the CHOP and the GEOMETRY nodes). Another important feature of this digital asset is the facility to pick the animations you like by displaying a univocal number for each one.

The casting digital asset uses a complete library of digital asset that is free for educational propose. This library was done by the Carnegie Mellon University Graphics Lab, and it’s opened to download in their website: http://mocap.cs.cmu.edu/.

In this library you can find a total of 2548 animations of human in different situations, from the one who is just standing up to the golf or soccer player. Those motion captures can be found in different formats, nevertheless the format that this digital asset uses is the biovision hierarchy (.bvh), that provides a skeleton hierarchy and the animation coordinates, this is a popular format because most of the software used in 3D animation handle this type of files. Houdini through the powerful command “mcbiovision” converts this .bvh into a .cmd (Houdini Script) and a .bclip (Channel Files). The .cmd provides the skeleton and the .bclip gives the animation for each bone of this skeleton.

Most of the knowledge used in this digital asset has been suggested by the vfxtoolbox developers with their tutorial called “Crowd” on their second chapter. They explain how to modify the loaded files and the way the animations are copied. The management of the files was guided by this tutorial; the design and the external communication of the digital asset are totally independent.

5.3 How to use

5.3.1 Inputs
A total of 5 inputs were selected to manage this digital asset. These inputs are used to modify the displaying animations and to select the liked ones.

Animations File Path: It is the file path for the folder that contains the motion capture library. It is necessary to specify the upper folder, i.e. the folder that contains the folders of the animation. In this case the default path is: /EDI/array2/CROWD_SIMULATION/COMMON/MOCAP. It’s important not to have spaces in the names of the folders, because it can cause confusion in the lecture of the files.

Index: It is used to change the casting batch, each batch represent the group of animations that are visible at the same time. The index is the number of the batch that is shown. It depends in the Batch size (also modifiable) and the total number of animations. The number of the first animation shown will be the number of this index multiplied by the batch size.

Batch Size: It is the number of animations that are shown at the same time. Inside the asset the batch size is the number of copies of the original skeleton and a auxiliary variable for the calculus of the index.
**Distance between animations:** It is the offset value in the x, y and z axis for the appearance of the copies, i.e. varying this parameter the animations will appear with a separation between them. The default value is 25 in the x-axis; therefore each copy will be spaced 25 points on the right from the previous one.

**Pick This:** We can say that this array is the outcome of this digital asset. Here the user type the numbers of the animations liked and automatically this digital asset read it and will be able to use them in other digital asset. (like Crowd Replication Static and Crowd Replication Dynamic). It’s important in how type the numbers, it is just the numbers separated by a space.

**Number of selection:** It is not an input! It’s an indicator of how many animations you have selected to your crowd system. Inside the digital asset beyond being an indicator, this represents another output to interact with others digital assets.

Depending on the performance of the computer that is being used to run this asset, the number of animations that can be seen at the same time with a real-time animation varies.

### 5.3.2 What is displayed?

In the Scene View Pane, all the navigations tools of Houdini can be used. Which are displayed from the digital asset are the skeletons and the index number in the upper part of them. This index number is univocal and it is associated just one by skeleton-animation. When you press the play button in the playbar (and with the real time option selected) you’ll be in grade to see the skeletons act.

![Figure 8 Casting - What is displayed?](image)

### 5.3.3 Uses, Limitations and Possible Modifications

The general propose of the casting digital asset is to bring the support and to lighten the crowd replication digital assets. Its function is to help and to reduce the time of selecting the animations, so it will be usable in different circumstances where it is needed to choose human animations, off course based in the Carnegie Mellon University Graphics Lab Library.

As it is said previously the casting digital asset doesn’t represent the final animation. You cannot attach an imported geometry to this files, i.e. always will be displayed the skeletons with the numbers. It is mandatory to be fully linked to existing files in the pc where the digital asset is used, if there is no files, it won’t work.

The first possible modification could be the possibility to attach an imported geometry in order to have a more tangible visualization. Sometimes it is hard to identify the movement of the skeleton because it is so tight and it gives some trouble to understand the movement.

Another modification could be adding the option to load other library, or to change the reference path to the files. Actually the casting digital asset works with the path and the library that is written by default. The big problem is that other library shouldn’t be compatible with the human skeleton given, so the change of the animations implies a change in the skeleton source.

Finally some others little modification on the controls can be added, like the scale, translation and rotation controls.
5.4 Design

5.4.1 Reading the motion capture data
After download the files from the Carnegie Mellon University Graphics Lab, the files must be converted to a format compatible with Houdini. The command mcbiovision is used to make the conversion of those files. To use this command it’s necessary to open a command prompt and expand to the directory of your .bvh file and write:

```
C:\File Path\> mcbiovision File.bhv
```

As it is said in the general description the treatment of the files has been guided by the “Crowd” tutorial of vfxtoolbox. To convert a big amount of files, vfxtoolbox has developed a free Digital Asset downloadable from their website (www.vfxtoolbox.com) called file_batch.otl, this asset execute the command for all the files within a folder, so it’s enough to put all the .bvh in the same folder and run this asset.

This action creates the File.cmd and the File.bclip needed to read the MotionCaptures inside Houdini. Then to open these files, inside Houdini open a text port and write:

```
>> source /File Path/File.cmd
```

At this point, there will be created: a Geometry Node (in the object level) that contains the skeleton hierarchy; and a Channel Operator Node (CHOP) that has the description of the movement. Inside the geometry node there are all the bones nested hierarchically, while in the CHOP node is necessary to change the channel file in order to load the .bclip.

This sequence will work for each one of the animations, but if the main objective is to create an asset where the user can change the source file in an easy and fast way, some changes inside the CHOP management are needed. The first thing to do is to change the reference in the bones translation and rotation parameters. The scope is to make this reference point to a changeable file reference which will be the CHOP channel file.

All the bones are nested hierarchically and any bone has translation information, all the translation information is contained at the top of the hierarchy that is the hip bone that will be called from now root. All the bones are packaged into a Houdini’s subnet by selecting them all and type shift+C on the keyboard, then an otl is constructed (right click create a digital asset) in order to can change the referred information. In the parameter interface of this mini-otl is created an operator path parameter in which will be received the information of the chop. So the referred operator in this parameter will be the output of the chop node.

On the chop node all the motion capture information is packaged in channels that will be called in the same way as the bones. In this order of ideas in the bones translation and rotation channels can be written an Hscript chop function to read this information.

```
chop(chsop("../chop")+"/$OS:$CH")
```

This function is copied in the translation vector of the root and in the rotation vector for all the other bones. The function chop read the information of the channel operator (read also by the chsop("../chop") that is the reference of the outer parameter) for the current operator ($OS) for the current channel ($CH). After this procedure the motion capture file is ready to be used.

5.4.2 Manipulating the CHOP
In order to read all the files without make this process to all the 2548 animations, the chop node need some changes. These changes involve the creation of a file master where all the names of the files .bclip will be stored in an array. Then by managing the indexes these files can be accessed from other operator. This file master is a file node type where the parameter interface is changed to have 3 parameters. The first one is a string that receives the path of the file’s container folder. The second parameter is another string that use a python function to read and store the .bclip file’s names.
These file names are taken and organized (indentifying the file extension .bclip using a pattern) into a string, each file separated from the previous one with a single space:

```python
import os
import re
dir=hou.pwd().evalParm('dir')
pattern='\.bclip'
list=[]
for root, dirs, files in os.walk(dir):
    for file in files:
        if re.search(pattern,file) is not None:
            full_path=os.path.join(root,file)
            list.append(full_path)
return ' '.join(list)
```

Loading the file names in this way will avoid looking for the files every time that the file is needed. Doing this procedure, the files are linked to expressions that will help to load them from remote nodes. It is important not to have files with spaces in the name, because it will cause a problem in the lecture of the files. The third parameter is an integer that uses another python function to read the number of files that has been read in the previous parameter:

```python
len(ch('file_list').split(' '))
```

This function just takes the information of the previous parameter and separate the string with the token \" \" space.

Also two auxiliary nulls are used to control how this information is accessed from the other nodes. The first one is called BATCH_CONTROL, and here is controlled the index of the animation to be displayed. The index of the animation is the position in the string of the file master, so the first file that appears in this file master will have the index 0, the second the 1, etc. To display the animations in groups the batch control uses the dimension of the group and a reference to the first index on this group, so two integer parameters are created. The other null serves to save the indexes of the animations that will be selected for the simulation; this null just has one string parameter that will appear also in the parameter interface, then this parameter can be read from external digital assets to communicate the selected animations.

In order to read the files without creating 2500 file nodes, are created around 30 file nodes that will have meaningful names (i.e. names with the pattern animations_NUMBER) in order to be accessed from the skeleton chop reference. These file nodes will have an expression to read the indicated animation from the file master.

**Channel file:** `ch("./FILE_MASTER/file_list").split(' ') [ch('index')]`

**Index:** `ch("./BATCH_CONTROL/index") * ch("./BATCH_CONTROL/batch_size") + hscriptExpression('opdigits("."))`

The index parameter serves an auxiliary to calculate the index for the meaningful name file. This index is calculating by multiplying the two parameters of the BATCH_CONTROL and adding the number of the meaningful name. Like the first file node (animations_0) will have the first specified animation, the second file node (animations_1) will have the second, etc. The channel file takes the name of the file in the position index of the file master array.

**5.4.3 Creating the copies**

The final step is to generate copies of the skeleton and assign each one a different ordered animation. To do this is enough to put a copy node and link the number of copies with the batch size number of the chop node. Also it's necessary to create a stamp variable that puts a univocal ordered value to each copy, like this is possible to assign a different animation to each copy. Nevertheless it is necessary to have a visual reference for the number of the animation, so is used the point number 0 of the skeleton to attach to him the index number. For this scope is created a font node which text will be an expression that reads the index of the animation that is being displayed:
Which is the same value as the index on the chop/file node, the only difference is that the opdigits function is replaced by the stamp value of the copy node. Then this font node is copied to the root point of the skeleton (obtained deleting the others with a delete node) and merging this result with the imported skeleton, before the copy.

The last thing to do is to refer the skeleton chop to the files with the meaningful names on the chop using the number of the copy. It is achieved with an Hscript expression:

```
../CHOP/animations`stamp("../clip_viewer/copy1","i",0)`
```

### 5.5 Performance Datasheet

Based on the number of points and polygons and the computer processor, the casting digital asset may vary in the results. In some computers you can execute and display 20 animations at the same time and it will look very good on real time, but in others you can execute and only achieve to display 5. So this is a reference to know what to expect and how many animations you will be able to see. Every single skeleton is composed by 592 points and 74 primitives; in addition this number is increased by the number of points and primitives from the numbers. A single number is composed around 50 points and 1 primitive, so it won’t represent a huge amount of extra computation. The highest time used in this digital asset is while the animation files are loaded into the CHOP node. Then the movements are computationally cheap, so the reproduction of the animation will be done in a fluid way.

Houdini provides a very interesting tool called “Performance Monitor” in which the user can select specific data to evaluate. In the case of casting.otl it has been chosen the Frame Time, the copy node time (where the skeleton is copied) and the loading time (i.e. the time in which the animation is read from the CHOP). Also is taken into account the average time for deforming a bone and the average time of copying the number into the skeleton.

All the tests have been done with the same conditions. The computer and the conditions of scale, frame and skeleton have remained the same for all the evaluated cases. The data taken from these tests is:

<table>
<thead>
<tr>
<th>Number Of Copies</th>
<th>Frame Time</th>
<th>Average Bone Time</th>
<th>Copy Number Time</th>
<th>Copy Skeleton Time</th>
<th>Chop Load Time</th>
<th>Total Load Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>44.98</td>
<td>0.07</td>
<td>2.91</td>
<td>170</td>
<td>100</td>
<td>170</td>
</tr>
<tr>
<td>10</td>
<td>105</td>
<td>0.07</td>
<td>1.5</td>
<td>200</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>15</td>
<td>142.59</td>
<td>0.07</td>
<td>1.5</td>
<td>450</td>
<td>100</td>
<td>450</td>
</tr>
<tr>
<td>20</td>
<td>176.58</td>
<td>0.07</td>
<td>1.5</td>
<td>450</td>
<td>150</td>
<td>450</td>
</tr>
<tr>
<td>30</td>
<td>263.75</td>
<td>0.07</td>
<td>1.5</td>
<td>700</td>
<td>200</td>
<td>700</td>
</tr>
</tbody>
</table>

**Frame Time:** The time that takes the computer to calculate and display an entire frame.  
**Average Bone Time:** The average time that takes to calculate a single bone.  
**Copy Number Time:** The average time that takes to attach the number to the skeleton.  
**Copy-Skeleton Time:** The time that takes the computer to make the copy of the skeletons.  
**Chop Load Time:** The time that takes the computer to load a single animation from a .bclip file in the Chop node,  
**Total Load Time:** The time that takes the computer to load an entire batch of the respective dimension.
From the data it is seen that the average time to read a .bclip increases when the number of copies increases too. This occurs because the information of the animations after read is stored into the RAM memory, causing a delay while allocating the data. Nevertheless, it only implies a death time while the files are being loaded.

On the other hand, the time that takes the processor to deform a bone remains the same, no matter the quantity of skeletons are on the scene. The same occurs with the first copy node, in which the number is attached to the skeleton. The time is the same for one digit’s numbers and two digits’ numbers.

6 CrowdReplication.otl
6.1 Introduction
The second part of this project regards to the implementation of the static version of the crowd simulations. The implementation of new Houdini’s functionalities and the application of the knowledge acquired in the previous part represent the basis for the construction of the digital asset. The main scope of this section is to deform the agents in a random mode, using the management of Houdini’s variables and attributes to apply textures and animations for creating independent and unique 3D human beings.

Theories from 3D modeling are used here. UV mapping, rigging and the final part of the implementation of motion capture are placed to construct every single human being. The UV mapping permits to create a 2D map of the geometry imported to simulate the humans in order to apply to him materials and make him looks like a real human being. This 2D map is exported as an image file in which the textures can be created by the superposition of photos or colors (in this case is colored because the resolution required is minimal because the LOD and the distance to the camera), then this constructed textures are saved into image files (is chosen the .jpeg format because it gives light files) and then with the creation of materials with UV texture maps are applied to the agents.

The process of matching the imported human geometry with the skeleton hierarchy provided by the motion capture is called rigging. It is used a modified copy of the skeleton to fit it with the geometry by changing the value of the translation and rotation components of each bone. Then with the help of Houdini’s tools the vertices that are in the proximity of bones are assigned to him, so if the bone is deformed, those vertices will be deformed as well. After this process to apply the animation of the motion capture to the geometry will represent the end of the motion capture pipeline. The result is the geometry moving in the same way that the captured human.

The end of the theoretical support for this section is the application of the Houdini’s tools learnt in the previous sections. This knowledge permits to assign in an ordered space, copies of the human geometry that act and look in different ways; to manipulate channels for animations; and to construct a package of nodes that will be manipulated from some external parameters that in the language of Houdini is called digital asset.

In the next paragraphs the way to manipulate and create this digital asset is explained deeply.
6.2 General Description
CrowdReplication.otl is a Houdini 10 digital asset, totally compatible with Houdini Escape and Master, designed to simulate a big quantity of people in a static situation, i.e. there is no translation, but there is a static movement of each person. Its main scope is to create crowd simulations in environments where there is no translation. This digital asset works by placing the humans into the environment and controlling many parameters such as density of population, location, orientation and number of animations.

Using a Motion Capture library for the movement of the agents, this digital asset is able to recreate realistic situations. Complemented with the Casting.otl; used to choose the movements to use in the animation (in the library there are around 2548 different human movements), the Crowd Replication Static digital asset reproduces the human behavior within some useful scenarios for static crowd simulations, like stadiums, concerts, conferences, etc. It receives inputs to set up the animation, the density of population, the number of polygons to use in the human geometry, the direction of the agents and the possibility to import external textures and geometries.

Automatically the Crowd Replication Static digital asset uses the selected animations, the default or loaded textures and geometries to create a crowd simulation. On this simulation each agent will act in a different way because each agent will have a unique parameter trio (texture, animation, and animation offset) depending of the univocal value of the point that provides Houdini.

6.3 How to use
As the Casting.otl do the CrowdReplication.otl and Casting.otl need to be pre-installed in the actual work, the steps to install a digital asset into the current work has been explained in the previous section.

With the digital assets installed, the first thing to do is to select the animations that will be useful for the actual animation. To do this task is necessary to use a casting digital asset (view Casting.otl section) that must be named casting1 (if it is the first node of this type in the actual work, it will be named like this by default). The only restriction for this selection is to choose animations without translation (i.e. no walking or running animations) because it won’t produce a realistic outcome. Automatically the crowd replication static digital asset takes from the casting digital asset the information concerning animation indexes and number of animations selected to assign it in a random way to the agents. It doesn't matter if there are a low number of animations selected; by shifting the animations a different number of frames (offset), it will produce a sensation of different moves. A recommended number of animations is between 15 and 50.

6.3.1 Inputs
To manage the digital asset there are twelve parameters. These input parameters are designed to have complete control of the animation, such as the density of population, the dimension of the agents, the minimum distance between them, the number of textures to use and the possibility to import files. To handle these parameters, the controls are in the parameter interface of the digital asset (see ¡Error! No se encuentra el origen de la referencia.), so it won’t be necessary to get into the sub-nodes to modify the animation. These parameters are:

![Figure 10 Crowd Replication Static - Parameters Interface](image)

Animations Directory: Due to the animations are external files, it is necessary to specify where those files are. It receives the path of the folder that contains the .bclip motion capture files.
LOAD POINTS: It is the main input of this digital asset. Here the points where the crowd will be placed are loaded as a geometry. This parameter is an operator path type, and receives a geometry filled with a high number of points. For example, in a stadium simulation, the stadium shape needs to be modeled (at least the stands) and within them is needed to make a scatter or a subdivision in order to generate points. Then these points are manipulated and used to produce the animation. It is recommended to pass more points that the wanted number, because then with the control of density this number of points is reduced significantly.

HUMAN SIZE: It is used to adjust the dimension of the agents to match with the dimension of the geometry that is loaded in the previous parameter. To see the dimension of the human the last input (Place/Points) must be equal to 0 and it's recommended to put a high number in the fuse Distance parameter to adjust it with a low number of agents.

Geometry Use: It's possible to load a model of the agent in a Houdini supported file extension. To do this action the parameter value must be one. Also the rigging must be done; to do this is necessary to right-click the crowd replication node and enable the "allow editing of contents" option. Then go inside and in the rigging node modify the value of rotation of the bones to have a right rigging. If this parameter is 0, the mcs_dude.obj will be used as default. (the mcs_dude.obj is a free-use human model donated by the motion capture society)

Geometry file: It is enabled when the previous parameter is set to one. This parameter receives the path of a 3D model file. This file must be supported in Houdini (several file extensions are supported in Houdini, depending of the version and the format some models can be read, for more information about this topic check the Houdini web site: http://www.sidefx.com/docs/houdini9.5/io/formats/geometry_formats).

Material Use: As default (value=0) are packaged into a SHOP node (that is inside the digital asset) 12 different materials referred to 12 jpeg texture maps images that are used randomly in the agents of the animation. These textures are done for the mcs_dude.obj file, if the model is changed, new texture maps must be loaded. If the value is one, a SHOP node with similar specifications to the default shop one must be loaded. Inside this SHOP node must be material nodes named with meaningful names with this pattern: human_NUMBER.

Material File: It is enabled when the previous parameter is set to one. This parameter is an Operator Path type, and receives the path of a pre-done SHOP node.

Fuse1 Distance: It is used to control the density of population of the crowd. When the geometry points loaded in the second parameter arrived to the asset, it is processed to be manipulated. This parameter controls the density of population by reducing the number of points based on the distance between them. The higher value, the less the number of agents.

Agents Direction: Another consequence of the treatment of the points is the capability to manipulate the orientation of the agents. The points normals can be modified to change the orientation of the agents. Handling these normals, you’ll be able to make the agents look to a single point in the space, to make all the agents look in the same direction or in the most advanced cases to write a function that will rule the orientation of the agents.

Polygons Percentage: In order to control the resolution of the agents, this parameter is used to reduce the number of polygons of the agents and therefore reduce the complexity and the cooking time of the scene. If the camera is very far, probably a low number of polygons can be used to achieve a nice view, but if the camera is near to the agents, almost the 100% of the polygons should be used. Nevertheless for applications where a high amount of humans will be used the camera is usually placed a considerable distance from the agents, so the percentage of polygons should be low.

Number of Textures: In the default SHOP there are 12 textures for the agents, but if you want to use less textures can be modified by changing this parameter. Also if a new SHOP node is loaded it needs to specify the number of textures that are used, in other case unexpected errors may occur.
Place/Points: To reduce the complexity during the set up of the scene and do changes easily this parameter permits to set the placement of the agents just by placing points. The display points option in the scene view pane must be enabled and this parameter must set to 1. When the value is turned to one, the points are replaced by the agents.

6.3.2 Procedure
After doing the selection of the animations, the next step is to design the geometry to provide the points position for the placement of the agents. It is important to design the shape and then with a scatter node or divide/subdivide nodes create as many points as needed. Maybe to have more points than needed could help to get a satisfactory animation, because with the fuse controls some points can be deleted. Finally, in the LOAD POINTS parameter is needed to specify the operator path to the output of the geometry, at this point those points arrive to the points treatment inside the digital asset.

These points treatment uses some user inputs that will imply a direct change in the outcome of the animation. The fuse1 distance represents the density of population control; the bigger is the value implies that the number of points/agents that will appear in the final scene will decrease. The other control used is the agent’s orientation that is restricted by the direction of the point normals. The normals keep a relation with the axis of the animation, if it's needed to rotate the agents within the XZ plane, the normal in Y will be 0 and the normals in X and Z will have a relation with the angle of rotation; the analog thing is applied for the other planes. Then the points are passed to the final stage of the animation.

This final stage is called placement because here is where the agents replace the points. The default geometry is loaded and then using a copy node the agents are placed in the scene. While the agents are being copied, the parameters to vary the animation, the texture and the animation offset are set to each agent using the stamp variables of the copy node. All those parameters will be assigned in a random mode, so all the agents will have different characteristics and there won't be any duplicated agents. In this stage some parameters to manipulate the final view of the scene are also set. These parameters are the scale of the humans, and the possibility to load personalized textures and geometries.

6.3.3 What is displayed?
The node that is displayed is the output of the placement node. There will be just the agents placed in the positions that were set in the imported geometry. The digital asset doesn’t contain geometries of the virtual or real environment, so it won’t be displayed. The default view is the points view mode that consists in the step before place the agents, i.e. all the modifications and calculus are done but bypassing the copy node that puts the human geometry replacing the points. Changing the parameter Place/points to 0 the bypass is ignored and the agents are placed in the scene.
6.3.4 Limitations
This version is focused on static crowds, so it doesn’t make animations of human that moves in space. For this scope the next section is the crowd replication dynamic. Another thing that it doesn’t do is to fix the rotation of the animations. In the library there are animations with different angles of rotation, so in the final animation (depending of the selection of animations) will appear rotated agents. The solution for this problem has been implemented in the crowd replication dynamic digital asset.

6.3.5 Possible modifications
Beyond the correction of the problems mentioned previously, some features can be implemented in this digital asset. The main feature is the possibility of guide a reaction of the agents, namely set a change in the animation of the agent by triggering an event, for example a goal in a soccer match for a stadium crowd. Other possible change is to be able to modify the behavior of the single agent during the animation. Actually, each agent runs one cyclic animation; the ideal case should be run a different sequence of animations for each agent.

6.4 Design
The Crowd Replication Static digital asset is composed of several nodes and hierarchies of nodes. Inside the main node there are 7 different sub-nodes that in turn are composed by basic nodes of the Houdini's respective type. In these 7 nodes of "second level" the main functions and features of the digital asset are set.

Within these nodes are divided the main functions of the digital asset. Some nodes are in charge of the adjustment of the geometry, others make the parsing of the motion capture files and others control the set up of the scene. Those nodes are communicated using some Houdini variables and expressions that can be written in Python or Hscript.

The main scope of this digital asset is the construction of the human copies, starting from the importation of the external files, arriving to the random assignment of textures and animations.
6.4.1 human_geometry

The function of this node is to manipulate everything respect the human geometry that is imported, starting from loading of the geometry and arriving to the texture mapping assignment. This node is communicated with other nodes in the digital asset to deform the geometry. First of all, the geometry is imported, scaled and moved to match it with an external skeleton to make the rigging. Then the proximity of the nodes to the bones is calculated in the "captureproximity" node. The reference for rigging is other node of second level that is called "rigging" in which a modified copy of the skeleton imported from the motion capture is used to attach the bones to the desired vertices. With the proximity done, it's possible to reduce the quantity of polygons used in the 3D model in order to reduce the complexity of the scene, Houdini provides the "Polyreduce" node and it is used and linked directly to the inputs of the digital asset where the user can select the percentage of polygons to use in the current scene. The next step is to assign the material to the geometry before deforming it. The materials are contained in the "SHOP" second level node and are named with meaningful names (following this pattern: NAME_NUMBER). To make the selection of the material random, in the material node the reference is composed by a string with the NAME_ plus an Hscript expression that reads a number from a copy node inside the placement node:

\[
(../../SHOP/human_\_\text{stamp}(".\_\_\text{placement}/\text{copy1}","k",0)),
\]

making in this way a random assignment of the materials to the agents. Finally the animation deformation is applied to the geometry. In a "deform" node another external skeleton in which the animation is loaded, is referenced. This skeleton is also a second level node called "skelet" and it loads the animations for the agents. At last, all the attributes are cleared and the geometry deformed is copied into a null that will be the output of the node.

6.4.2 rigging

This node is used to complete the rigging. It is referenced in the human_geometry/captureproximity node and consists in a copy of the skeleton that is used in the motion capture, but instead of being animated is fixed to fit with the geometry. Every bone is transformed and rotated in a way to match with the position of every part of the imported geometry. Then the geometry is scaled and translated to fit with the modified skeleton. This proximity must be calculated in a single frame, for this reason is used an unanimated copy of the skeleton that will have the same translation and rotation values during the animation sequence.

6.4.3 SHOP

All the materials for the agents are packaged in a SHOP node. This SHOP node is at object level but can be replaced at SHOP level. The node consists in 12 material nodes that have meaningful names to randomize the assignment of them. These materials use external color maps (.jpg images) created after
the uv projection of the geometry. The images are not necessary photorealistic and are done in a light format because in a crowd replication the level of detail needed isn’t big.

6.4.4 Skelet
When the motion capture is imported into Houdini, there are two files, the .bclip and the .cmd. The .cmd is the script of how the skeleton is composed. It is loaded using the textport of Houdini, writing the reference to the file:

```
>> source ../My_path/my_file.cmd
```

This command will load the skeleton into the object level. Every bone is a node nested hierarchically where the top of the tree is the root (the hips) and the leaves are the tips (arms, legs and head). In the nodes the information of rotation and translation are packaged in vector parameters and referenced to the motion (rotation for all the bones and joints and translation for the root). This motion is loaded into the CHOP node where the .bclip is imported. This "skelet" node has the property of being capable of load the animations in the same skeleton by changing the referenced CHOP in an external parameter. The rotation and translation information for all nodes are referenced to the CHOP by using a Hscript expression:

```
chop(chsop("../chop")+"/$OS:$CH")
```

where

- `chop()`: Evaluates a channel within a CHOP at the current time.
- `chsop()`: Evaluates the parameter at the current time as a node path string.
- `$OS`: Operator String. Contains the current operator’s name.
- `$CH`: Current channel name.
- `../chop`: is the path of the parameter that references the CHOP to load. This parameter is in the "skelet" interface.

6.4.5 CHOP
The animations imported from the motion capture must be treated and modified to have a realistic outcome. In this node the movement channels are read and set up to make the animations cyclic and to have a specific frame range. First of all, the animations must be imported, and to reduce the complexity of the scene and import the animations once is created a FILE MASTER. In this file master is specified the path of the folder that contains the .bclip files. Using a Python function, these file names are taken and organized (identifying the file extension .bclip using a pattern) into a string, each file separated from the previous one with a single space:

```python
import os
import re
dir=hou.pwd().evalParm('dir')
pattern='\.bclip'
list=[]
for root, dirs, files in os.walk(dir):
    for file in files:
        if re.search(pattern,file) is not None:
            full_path=os.path.join(root,file)
            list.append(full_path)
return ' '.join(list)
```

Loading the file names in this way will avoid looking for the files every time that the file is needed. Doing this procedure, the files are linked to expressions that will help to load them from remote nodes. It is important not to have files with spaces in the name, because it will cause a problem in the lecture of the files. The other part of the CHOP represents the animations nodes where the animation will be modified and fixed to be loaded by the skeleton. Inside this nodes (that have meaningful names) is processed the information from the casting.otl inputs. First of all, it takes the index number of the animation and reads from the file batch the file .bclip to use.
This python function is used to read the information from the casting.otl input. opdigits() is a function that reads the number from a meaningful names. So the first number in the array "pick this" of casting.otl will be loaded into the first animations in the CHOP. The index is a univocal value for each .bclip file.

Then in the channel file, using the response of this function and the list contained in the file master is loaded the .bclip file. After that this information is passed to the skeleton and the animation is ready to be used.

When the file is loaded, we need to fix the animation. For this scope three nodes are used. The first one is the trim node that serves to cut the animation at the beginning and at the end. The second one is the cycle node in which we can specify how many times the animation will be repeated and the third one is the shift node that is used to set the local start frame to play the animation.

For this scope, the motion view channel has been utilized. In this motion view the cinematic curves for all the joints of the skeleton can be visualized and modified, however because the scope of this digital asset is to simulate a static crowd and the only joint that represents a translation are the hips, the interest for handling is centered in this node. To avoid the translation of the entire skeleton, it's necessary to delete the components of the cinematic of the hips in x and z. With just the Y component in the motion view there will be just one curve to manipulate.

As said before, the first thing to do is to trim the animation, namely to set the starting and the ending point of the animation by moving the node parameters. The Y-translation component only represents a vertical movement, so this curve will be represented with valleys and crests. The main idea is to make an even number of crests and to make similar the value of ending and beginning for blend the movement that will be cycled.

The second step is the cycle node. Here the number of times that the animation will be played is set. The decision is based in the frame range that is needed in the entire animation and the value of the random offset that will move the beginning of the animation backwards. With the cycle set as the default view in the motion viewer, the trim values of start and end can be changed to have a smooth curve and as consequence a more realistic outcome.

Because the animation has been cut in the start and the end and as consequence there will be a frame range in which there is no animation, it’s necessary to move the animation backwards to be played from the frame 1. The node shift provides a tool to do this thing just by referencing the beginning of the animation with the trim start value with an Hscript expression. This expression reads the value of the node in the input.

- chops(opinputpath(".",0))

Where, 

chops():Returns the start index of a CHOP
opinputpath(".",0): Returns the full path of the node connected to a given input.
And the minus means that the animation will be moved to the left.
Beyond shift the animation to the new frame 1, also in this node an offset (in frames) for the animation can be set. To make this value univocal for the agents, it is set randomly when the agents are placed using the stamp variables.

![Figure 18 Crowd Replication design](image)

### 6.4.6 points_treatment

Before running the animation, a final step has to be done. The points received from the external (to the digital asset) geometry must be treated and fixed to be used. In this node, the points are received using an “object merge”. Then the points enter into a fuse node. Its function is to reduce the density of points in a zone. A threshold for the distance between points is chosen externally (by user input) and the points that are nearer than this threshold are fused. The orientation of the normals of the points will influence the orientation of the agents, so a point node to add a normals’ control is set. Both controls: normals’ value and fuse distance are linked to a user inputs.

### 6.4.7 placement

The final step is to copy the agents to replace the points. For this scope we need as inputs the output of the points_treatment and the output of the human_geometry nodes. The agent’s geometry is manipulated with a transform node in which the uniform scale is linked to a user input. Then the points are replaced with the agents with a copy node. In this copy node some stamp variables are created in order to make the agents unique. These stamp variables set the animation and its offset, the texture and the position and are linked to other parameters in other nodes to load different files as is shown in the figures below:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Expression</th>
<th>Linked to</th>
<th>Use</th>
</tr>
</thead>
</table>
| I       | round(fit01(rand(SPT)*1.1,0,ch("/obj/casting1/NA")-1)) | Skelet/chop
Chop/animations
Casting1/number of animations | This value represents the meaningful number of the CHOP/animations node that will be assigned to the agent depending of the number of the point (SPT). It can take a number between 0 and the number of animations selected in the casting.otl minus one. |
| K       | round(fit01(rand(SPT)*1.1,0,ch("..././number"))) | Human_geometry/material
Input: Number of materials. | This value represents the meaningful number of the SHOP/human node. It is used to assign different materials in a random way. The number range is between 0 and the number of materials that the user set in the parameter interface (default value = 12). |
| W       | round(fit01(rand(PT), -300, 0)) | Skelet/offset
Chop/animations/shift | This value is the number of how many frames will be moved the animation before it starts. The frame range is (-300,0) |

The expressions are similar form because are based in the univocal number for point that assigns Houdini. There will be no points with the same number (SPT) so it’s warranted that there won’t be identical agents. The expressions are composed by 3 functions:
Round: to convert a float value into integer.
Fit01: to convert a value between 0 and 1 to values in a specified new range.
Rand: to create a random number between 0 and 1 for an input.

Then to reduce the complexity of the asset while the animation is being set up, a switch node gives the option to visualize the points or the agents. If the view is set to be the agents view the changes won’t be on real time and every change that we want to make will take several minutes, but if there is the points view all the changes will be done in real time and the calculus of the final animation will be done just once, but the display points option in the scene view pane must be enabled. This switch is connected to a null node that will represent the output view of the entire digital asset.

### 6.5 Performance Datasheet

To evaluate the performance of the digital asset it is necessary to confront different number of agents in a scene with the time of calculation, the time that uses the copy node to reproduce the crowd and many other times spent by each node; varying the percentage of polygons per agent and the visualization mode that are used. Using the Performance Monitor of Houdini, the information is taken and resumed in the following tables and graphics:

**Human Geometry:**
Case 1: Polygons: 4%; Wireframe
Case 2: Polygons: 10%; Wireframe
Case 3: Polygons: 10%; Smooth Shaded
Case 4: Polygons: 4%; Smooth Shaded

The time values presented on the table below are calculated once by frame. The Frame Time represents the total value that the computer spends in calculate and display one frame. The Placement Copy Time represents the time that takes the copy node in the Placement subnet, namely the time that take the computer in produce the copies of the human geometry.

<table>
<thead>
<tr>
<th>Agents (number)</th>
<th>Case 1</th>
<th></th>
<th></th>
<th>Case 2</th>
<th></th>
<th></th>
<th>Case 3</th>
<th></th>
<th></th>
<th>Case 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>95,48</td>
<td>4,95</td>
<td>72,95</td>
<td>6,47</td>
<td>99,4</td>
<td>6,37</td>
<td>72,92</td>
<td>6,57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>222,23</td>
<td>23,43</td>
<td>251,83</td>
<td>37,55</td>
<td>257,6</td>
<td>41,16</td>
<td>216,55</td>
<td>25,86</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>507,22</td>
<td>86,81</td>
<td>635,82</td>
<td>154,11</td>
<td>626,35</td>
<td>165,03</td>
<td>512,97</td>
<td>97,69</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>1050</td>
<td>259,05</td>
<td>1370</td>
<td>490,37</td>
<td>1420</td>
<td>514,4</td>
<td>1080</td>
<td>284,98</td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>2320</td>
<td>809,42</td>
<td>3360</td>
<td>1650</td>
<td>3480</td>
<td>1600</td>
<td>2450</td>
<td>872,04</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


From the data can be noticed that the parameter that affect in a higher way the time of calculation is the number of polygons of the geometry. From the first table that the number of polygons are duplicated passing from 4% to 10%, however the copy node requires more than two times to complete its job, but on the frame time can be noticed that the difference of both (4% and 10%) is almost the same difference that is seen on the copy node, that shows that all the other operations remains with the same complexity, no matter the number of polygons. Also can be noticed that the difference between the visualization wireframe and smooth shaded in question of time increase with the number of agents, nevertheless doesn’t represent a significant value in order to the benefits that carries a visualization smooth shaded.

The values on the table below are computed once by copy (human geometry). Those values presented are the average time to deform each bone of the skeleton, the average time for the entire
Human_geometry subnet, the average time for the placement subnet nodes (except the copy node) and the average time for the CHOP process of trim-cycle shift.

<table>
<thead>
<tr>
<th>Case</th>
<th>Bone time</th>
<th>Human_geo placement</th>
<th>Chop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>0.07ms</td>
<td>1ms</td>
<td>0.5ms</td>
</tr>
<tr>
<td>Case 2</td>
<td>0.07ms</td>
<td>1.3ms</td>
<td>0.9ms</td>
</tr>
<tr>
<td>Case 3</td>
<td>0.07ms</td>
<td>1.3ms</td>
<td>0.8ms</td>
</tr>
<tr>
<td>Case 4</td>
<td>0.07ms</td>
<td>1ms</td>
<td>0.5ms</td>
</tr>
</tbody>
</table>

Nevertheless, this digital asset provides an excellent option to reproduce crowds with a reasonable complexity cost. Another vantage is managing the LOD (level of detail) when are needed more agents, there will be probably needed a reduce resolution, and when is needed a high resolution, there will be probably needed less agents. It is also remarkable that this digital asset doesn’t use rigid bodies or particles, so it can be run from Houdini Escape.

6.6 USES

The Crowd Replication tools can be used in many different situations, from a concert simulation to the use of the agents as extras in a scene. Within these possible applications, the stadium crowd is the most demanded because in several countries sports represent the main channel to communicate a message.

The first real application of the developed crowd replication tools has been the TIM Falling Star spot in which is showed TIM as sponsor of the Italian Serie A League. In this spot a fake stadium ought to be filled with fake people. In order to fill the stadium around 15 real people were recorded to be placed in certain positions of the stadium. The rest of the stands have to be filled with 3D human-like geometries. The selected tool to do this is the crowd replication static.

Starting from a San Siro stadium photo the stands of the south curve has to been modeled and filled. Placing the photo as a background into the Houdini's viewport, the camera has to be set as similar as the camera used for the photo, moving the focal distance, the position of the camera and the resolution. Then the stands were modeled using the different views and adjusting them with the camera's one to fit with the dimensions of the real stadium. These stand representations are the input geometries for the digital asset. Every single stand is a square-like geometry with a scatter or subdivision to increase the number of points in a notable way. The number of points has to be high because inside the digital asset can be found a density control where the number of points can be reduced by varying the parameter of a fuse node.

![Figure 23 Stands Modeling](image)

On the other hand the animations selection has to be done using the casting digital asset. After this process a total of 14 animations were selected to represent the people. The pattern to select the animations was the need of static people but euphoric movements, namely people that must be always on the same place but with a very notable movement such as hands up or hands and head movements. Those animations were then cycled on the CHOP node inside the digital asset.
With this two inputs ready to be used (the stadium geometry and the animations selected), the appearance parameters need to be set. These parameters involve the agents direction (they have to face towards the camera), the textures (there were used 12 different textures for the agents), the human size (for the animation was needed the 75% of the original dimension), the density of the agents and the level of detail for the scene. Regarding the density it is chosen a relative low density to achieve a good mix with the recorded material of the real stadium people.

Then some rendering considerations have to be taken. Because the whole scene involve the complete south curve of the San Siro Stadium more than 5000 fake agents are needed, so a fragmentation of the scene has to be done. In the final scene this fragmentation shows six different pieces of the curve and as consequence six different rendering sequences. Also a light animation was required for the scene, so two spot lights at the top of the stadium were placed and they were animated using key framing of the light intensity. The rendered sequence with the agents, lights and stands are exported as .sgi sequences. Then the images are composited using Apple Shake 4. In the definitive scene can be found: the matte painting of the stadium, the animated agents from the digital asset color corrected to fit with the real agents, flash lights and stadium lights introduced in Shake and the chroma-key of the scene recorded.

This spot involve several other visual effects done by the EDI staff, like the soccer ball modeling, the broken net animation, the chroma-key and matte painting for most of the scenes.

Finally the spot can be watched during the 2010-2011 season of the Italian Serie A League matches.

7 CrowdReplicationDyn.otl

External Files Needed:

**Motion Capture library:** Files with .bclip and .cmd extensions.

**Human Geometry:** mcs_dude.obj or other supported geometry.

**OTLs needed:** skeleton.otl, MoCapCycle.otl, casting.otl.

**Textures:** Texture maps images.

7.1 Introduction

The third section of the project represents the implementation of a brain for the agents in order to create a simulation of a dynamic crowd. All the information of the previous sections (casting and CrowdReplication) is used and complemented to create this third digital asset. Beyond the topics treated in the previous section, here is taken theoretical support from the artificial intelligence techniques and the behavioral modeling implementation.

The introduction of the translation of the agents represents a sequence of problems that have to be solved. These problems involve the management of the collisions with the other agents and with the obstacles placed in the virtual world; fixing the animations to be always in the same direction controlling the information of the translation channels; making the agents take inputs from the ground to follow it; and the creation of the entire virtual world.
The argument that is highly treated in this section is artificial intelligence and simple neural networks design. The agents take information from the environment (input) to compute it (neurons) and produce a response according to the situation (output). No matter if the environment changes the agents will always response according to the actual situation. The cognitive approach is taken into account to perform an adapt brain for the simulation. The computation of the received information has a mathematical and physical theoretical support in which can be found cinematic treatment of channels and parameterization of simple curves to describe evasion movements.

The main argument regarding the implementation of these actions in the Houdini’s environment is the use of point attributes. The digital assets have been created without the use of particles or rigid bodies so can be run with Houdini Escape that is cheaper and sometimes faster than Houdini Master. To achieve the complete perform of the functionalities wanted, the management of points attributes in every moment before being replaced with agents is fundamental to the objectives set. The management of those attributes permits to create variables that can be computed and transferred in a comfortable way.

All over this section is explained how to use the digital asset and how was constructed.

### 7.2 General Description

CrowdReplicationDyn.otl is a Houdini 10 digital asset, totally compatible with Houdini Escape and Master. Its main scope is to create crowd simulations in dynamic environments. Based on the Motion Capture Library that provides realistic human movements, this digital asset reproduces the human behavior in situations where a translation is needed, taking into account factors that may affect the similarity with the real life, such as obstacles, interaction between agents and interaction with the world.

In a similar way to the crowd replication static version (see the previous section), the dynamic version uses the casting.otl to make the selection of the animations that will be used. The most important feature regarding this aspect respect the static version, is the animations rotation fixing. It is possible by reading the channel information and manipulating it from the object level. The result of this procedure is that the agents, no matter their rotation, will always move into the local z-axis with the original animation speed. Nevertheless, one problem persists from the static version and it's the fact that to fix the animations with the trim-cycle-shift nodes before the set up of the scene is needed. The way the textures, animations and position (for static version animation position, for dynamic version initial position) are assigned to the agents remain the same from the static version, so is warranted its random nature and that all the agents will be different.

The agents are placed into a pre-created world and are able to interact with it to have a mutual interaction and follow a linear route avoiding collisions with two different types of obstacles. Also the agents are in grade to follow the terrain, it doesn't matter if there are irregularities and mountains, the agents will rise or go down depending on the situation. Because the agents receives information from the virtual world (understood as obstacles, other agents and ground), specify the shapes and dimensions is needed. With this information used as input for the agents, an artificial intelligence-based system rules the behavior and the criteria to take decisions about an action of the agents following the principles of the Behavioral Modeling.

A complex user parameter interface has been designed to be able to set up the scene in a complete and easy way. This user interface also permits to have a visual guidance to set up the world and to change the human behavior (changing the level of tolerance for avoiding objects and agents) that can be deactivated in any moment.

This digital asset is ideal to simulate dynamic situations for human crowds like manifestations, marathons, wars, concerts, etc. Also other application should be humans that are in the background of a scene, namely the extras.

### 7.3 How to use?

The first thing to do is to install the casting.otl and the CrowdReplicationDyn.otl inside the current work of Houdini. The procedure to do this is explained in the casting.otl section. After the installation
of the digital asset it's needed to add one node of each type into the network view, and start with the selection of the animations. In this case there is no restriction in the selection of the animations, because this digital asset supports static and dynamic animations and the rotation problem has been resolved. As in the static version, the casting node must be named casting1, because the path to the inputs of this node is linked within the CHOP node.

Now the Crowd Replication Dynamic digital asset shows the default view and the parameter interface. In the default view there are loaded a default ground geometry done with a deformed plane, one type of objects with a random placement (using a scatter of the ground geometry) and 30 agents placed using a scatter of a scale down version of the ground geometry. Also the visual guidance is displayed by default (to use this guidance is recommended to use the wireframe view).

In the next paragraphs there is an explanation of how the visual guidance works, and a complete description of all the parameters that are contained in the Digital Asset's parameter interface.

7.3.1 Visual Guidance

In this visual guidance are displayed: 1. a color code for the agents’ actions. 2. An area of placement of the agents. 3. The obstacle avoidance paths.

The color code for the agents’ actions is the color of the agent when is happening something in the scene and are used four values to identify the possible actions:

- **WHITE**: If the agent is doing his normal action (the linear path)
- **RED**: If the agent is avoiding another agent, so they are modifying their linear path adding an evasion curve.
- **GREEN**: If the agent is avoiding an obstacle taking the right side of the avoidance ellipse.
- **BLUE**: If the agent is avoiding an obstacle taking the left side of the avoidance ellipse.

The area of placement of the agents, to set the initial positions, is displayed as a light blue area, using the parameter interface is possible to modify this area and the changes will be applied on real time to the visual guidance.

Also are displayed the obstacles avoidance paths. These paths are ellipses that will be taken if the agents are getting closer to an obstacle. These ellipses are modifiable in the parameter interface, in the Human Behavior pane.

7.3.2 Parameter Interface

The parameter interface designed to this digital asset involves everything concerning the animation. In contradiction to the static version where the world geometry is designed outside the asset, the entire environment is designed inside. The reason is that the agents must receive the information as input to the decision of their actions, and loading an external world can cause serious problems with offsets. Nevertheless there is an option to load a ground shape and transform it, as well for the obstacles, its shape also can be imported and their placement can be designed using a curve.

The parameter interface is divided in 4 tabs. The first one is the default transform tab, where the entire digital asset can be transformed following the Houdini parameters. The second tab is the world set up tab which is also divided in other 3 tabs: Agents, Obstacles, and Ground. The third tab is called Human Behavior and there are all the parameters that will control the avoidance of the agents. The last tab is called Cache Animation and permits to calculate the geometries of the world and of the humans separately to reduce time while doing tests. Outside these parameters there is the option for deactivate the visual guidance.

7.3.2.1 World Setup

In this tab there are all the parameters that will modify the view of virtual world, like scaling parameters, setting positions, or importing external geometries for ground or obstacles, etc. In order to have an ordinate and easy handle of the parameters, this tab is divided into other 3 sections, each one for the one part of the animations, namely Agents, Obstacles and ground.
7.3.2.1  GROUND

**Ground**: Default/Load: This parameter serves to select if use the default ground for the animation or to import a geometry that can be modified with the following parameters. If the value of this parameter is 0, the default ground is loaded. This default ground is a plane deformed with a mountain node, so it's a very irregular terrain. But if the value is 1, the following parameters came enabled to load and modify the geometry.

**Ground path**: It is an operator path type parameter. If the previous parameter value is 1, this parameter is enabled to load the geometry that will be the ground. This geometry must be created in the current work of Houdini.

**Ground Scale**: To make the ground match with the dimension of the rest of the components of the animation such as obstacles and agents, this parameter scale the imported geometry. Also this parameter is enabled if the first parameter is set to 1.

**Ground Rotate**: This parameter has the same scope than the previous parameter, and it is to modify the imported geometry to make it match with the virtual world. The function of this parameter is to rotate the ground (just the ground without obstacles or agents). Also this parameter is enabled if the first parameter is set to 1.

**Ground Translate**: Like the previous two parameters, this parameter also modifies the ground. It translates the ground to adjust the position respect the others objects. Also this parameter is enabled if the first parameter is set to 1.

**World scale**: Contrary to the previous parameters, this parameter will modify everything except the agents. It represents a global scaling for the environment where the agents are.

7.3.2.1.2  OBSTACLES
On this digital asset, the agents are able to avoid 2 different types of obstacles. For each one of them there are some parameters to modify their setup in the scene:

**Number of obstacles:** It is the number of obstacles of the first type that will be placed in the world. This number could vary in the final animation, because to prevent an error in the first frames the obstacles that are near to the initial position of the agents are deleted.

**2 Number of obstacles:** It is the number of obstacles of the second type that will appear in the scene. Also some of them are deleted in the first frame to prevent errors.

**Obstacles:** Load/LSystem/Tube: This parameter is used to select the geometry of the first type of parameters. By default are created an L-System and a Cylinder. If the value of this parameter is 0, a geometry can be loaded in the next parameter. If the value is 1, there will appear obstacles with an L-System form. If the value is 2, there will be cylinder obstacles.

**Object path:** (Geometry): It is used to load a designed geometry to the environment. It is an operator path, so the geometry must be designed into the current work of Houdini. Also can be imported from an external file and converted into a geometry.

**1 Default/Load/None:** This node serves to control the placement of the objects. If the value is 0, the asset will use the default placement that is a point’s scatter of the ground geometry. If the value is 1, a placement can be designed and imported in the next parameter. If the value is 2, there won't be any obstacles of this type in the scene.

**1 Objects path:** (placement): It loads a designed placement for the first type of obstacles in the scene. A recommended procedure to use an external placing is to design a curve in the top view, where each point will represent one placed obstacle. Then this curve is loaded on this parameter.

**New Type of Obstacles**

**Type of obstacles:** Load/LSystem/Tube: It is used to select the geometry of the second type of obstacles. See Obstacles: Load/LSystem/Tube.

**LOAD OBJECTS:** This parameter loads the geometry of the second type obstacles. See Objects path (Geometry).

**2 Default/Load/None:** This is the placement control of the second type. See 1 Default/Load/None.

**2 Object path:** (placement): It loads the external placement for the second type. See Objects path: (placement).

**7.3.2.1.3 AGENTS**

![Figure 27 Agent’s setup parameters](image)
**Placement: default/load:** This parameter is used to indicate what to use for the agents placement. If the value is 0, the default placement is used. This default placement is an scaled version of the ground with scattered points. If the value is 1, an external geometry for place the agents can be imported.

**Placement scale X:** In order to cover the width of the ground with agents, this parameter scales the placement geometry in the x-axis. It will be applied to the default or to the imported geometry.

**PLACEMENT: (load):** If the first parameter of this tab is set to 1, this parameter is enabled to load the geometry for the placement of the agents.

**Number of Agents:** It is the number of agents that will appear in the scene. Some agents can be deleted in the first frame to prevent errors. When agents are loaded, the digital asset sees the distance between agents and if the distance is less than the threshold set (in the human behavior tab), one of the agents will be deleted.

**Human Scale:** It is used to scale the agents. In the ground tab, the parameter world scale scales everything except the agents; this control is done for this propose.

**Human Geo:** It's possible to load a model of the agent in a Houdini supported file extension. To do this action the parameter value must be one. Also the rigging must be done; to do this is necessary to right-click the crowd replication node and enable the "allow editing of contents" option. Then go inside and in the Rigging node modify the value of rotation of the bones to have a right rigging. If this parameter is 0, the mcs_dude.obj will be used as default. (The mcs_dude.obj is a free-use human model donated by the motion capture society).

**Load Geometry:** It is enabled when the previous parameter is set to one. This parameter receives the path of a 3D model file. This file must be supported in Houdini (several file extensions are supported in Houdini, depending of the version and the format some models can be read, for more information about this topic check the Houdini web site: [http://www.sidefx.com/docs/houdini9.5/io/formats/geometry_formats](http://www.sidefx.com/docs/houdini9.5/io/formats/geometry_formats)).

**Material: Default/Load:** As default (value=0) are packaged into a SHOP node (that is inside the digital asset) 12 different materials referred to 12 jpeg texture maps images that are used randomly in the agents of the animation. These textures are done for the mcs_dude.obj file, if the model is changed, new texture maps must be loaded. If the value is one, a SHOP node with similar specifications to the default shop one must be loaded. Inside this SHOP node must be material nodes named with meaningful names with this pattern: human_NUMBER.

**Load Material:** It is enabled when the previous parameter is set to one. This parameter is an Operator Path type, and receives the path of a pre-done SHOP node.

**Polygons:** In order to control the resolution of the agents, this parameter is used to reduce the number of polygons of the agents and therefore reduce the complexity and the cooking time of the scene. If the camera is very far, probably a low number of polygons can be used to achieve a nice view, but if the camera is near to the agents, almost the 100% of the polygons should be used. Nevertheless for applications where a high amount of humans will be used the camera is usually placed a considerable distance from the agents, so the percentage of polygons should be low.

### 7.3.2.2 Agent Behavior

In this tab there are all the parameters that will affect the human animation, namely all the parameters that won't modify the virtual world view, but during the animation will affect the behavior of the humans. These parameters serve to load the animations and to handle the avoidance.
Animations Directory: Due to the animations are external files, it is necessary to specify where those files are. It receives the path of the folder that contains the .bclip motion capture files.

Min Human Distance: This value is the distance threshold for the first frame between agents. Agents that are closer than this threshold will be deleted.

Human-Object Initial Distance: This value is the distance threshold for the first frame between agent and object. If the distance is less than the threshold, the object will be deleted.

Human Object Avoidance: For the rest of the animation, this will be the minimum distance in which the agent will divert his way to avoid the object. If the distance with the object is less than this value, in the visual guidance the agent will turn blue or green, depending on the x-axis position.

Avoidance X-Radio: Because the avoidance of the obstacles is an elliptic divert of the lineal part, this parameter will represent the maximum x-axis divert, or the minor axis of the avoidance ellipse. This parameter will be applied in the first type of obstacles.

Avoidance Z-Radio: This parameter will represent the maximum z-axis divert, or the major axis of the avoidance ellipse. This parameter will be applied in the first type of obstacles.

Avoidance X-Radio: See Avoidance X-Radio. This parameter will be applied in the second type of obstacles.

Avoidance Z-Radio: See Avoidance Z-Radio. This parameter will be applied in the second type of obstacles.

Human Avoidance Distance: This parameter represents the minimum distance between agents during the animation. In the visual guidance, if the distance between agents is shorter than this value, the agents will turn red, and will start to make a mutual avoidance.

Human Avoidance Radio: Like the agent-obstacle avoidance, the agents avoid each other by diverting from their original linear path, following an imaginary circle around the other agent. This parameter represents the radius of this circle.

7.3.2.3 Cache animation
In this tab the animations can be calculated and saved on .bgeo files. It is separated into two parts, one to indicate the animation parameters for the world and the other the parameters for the agents.

Render Button: When is pushed start to calculate the animation within the frames and the values described below.

Valid Frame Range: It serves to indicate the frame range to calculate the animations.

Start/End/Inc: It indicates the start frame, the end frame and the step for the calculation.

Output File: Indicates the output file where the calculation is saved.
7.3.2.4 Display Parameters

**Display Help:** This parameter serves to activate or deactivate the visual guidance. 0 Deactivate, 1 Activate, Default=1.

**Display Ground:** This parameter serves to visualize or not the ground. 0 Deactivate, 1 Activate, Default=1.

**Display Obstacles:** This parameter is used to visualize the entire environment: the obstacles and the ground. 0 Deactivate, 1 Activate, Default=1.

7.3.3 Procedure

With the default view put in the scene, the next step is to manipulate the parameter interface to achieve the desired animation. The first thing to do is to setup the environment, using the World Setup Tab, before placing agents. A suggested order is to begin with the ground, continue with the obstacles and finally handle the humans. This order permits that you can use the ground geometry to place the agents and the obstacles.

7.3.3.1 Setup the Scene

As shown in the parameter interface for ground (World Setup/Ground) the parameters to handle the ground are deactivated if the default ground is used. But if an imported geometry is used, the parameters permit to have a desired ground. The dimension of the imported geometry should be similar to the dimension of the default geometry, like this the ground will match automatically with the dimension of the other 3D objects in the scene.

In the suggested pipeline, the next step is to place the obstacles. Using the default placement, the obstacles will be placed randomly all over the surface of the ground. If you want to place the obstacles by yourself it is recommended to use the top view and design the curve. It doesn't matter the Y axis in this curve because then during the interaction stage, the objects are lift to calculate the agent's interaction and in the placement stage the obstacles are put at the level of the ground.

The third step is to place the agents. This procedure is similar to the procedure done for the static version; the only difference is the possibility to modify the placement zone inside the parameter interface. In the parameter interface is set a default placement zone that is a scaled version of the ground. The main modifier is the X-Axis scale which is put to cover all the width of the ground. If the placement used is the imported one, the procedure is the same than the static version. (See CrowdReplication.otl). Also the visual guidance provides a help to achieve a right placement.

7.3.3.2 Controlling the Agents

After the setup of the scene, it's necessary to vary some parameters that will rule the agents' actions. These parameters are in the Agent Behavior tab of the parameter interface and their main propose is to handle the interaction between agents and the interaction human-obstacle. These parameters represent the input of the Artificial Intelligence system designed for the agents, and are the variable thresholds for the distance with the others 3D objects. Because the dimension of the objects is different, the threshold for a human should be smaller than the threshold for the obstacles. The visual guidance provides a real time visual help to handle this parameters. The color of the agents (as described before) represents that the agent is changing their normal path and is into a zone of possible collision. The ellipses represent the zone of a possible collision with an obstacle.

7.3.3.3 Selecting the view

Finally using the display parameters, you are able to select what to see. Note that the default obstacles are placed to have a reference of the dimension of the obstacles, they have no textures and aren't modeled. The display options are shown in the next section.

7.3.4 What is displayed?

Dependent on the choice of the display parameters, there are some view modes that can appear into the scene view pane. By default the only 3D objects that will be always visible are the agents (but if you put in the parameter: number of agents=0, there won't be agents). The other 3D objects can be disabled by handling the parameters. The possible views are:
**Help Mode:** Here the guidance for placing the agents and ruling their behavior are displayed. This view is reached by setting 1 in the display help parameter.

![Help View](image)

**Entire World Mode:** This happens when the display obstacles parameter is set to 1 and the display help parameter is set to 0. Here all the 3D objects in the scene will be displayed.

![Entire View](image)

**Non-visible obstacles mode:** The display obstacles and display help parameters must be 0 and the display ground must be 1. Here are displayed the agents and the ground.

![Non Visible Obstacles View](image)

**Just Agents mode:** All the display parameters are set to 0. And in the View Pane only will appear the agents.

![Just Agents View](image)

### 7.3.5 Limitations

Each agent only can take one animation, so there is no speed changes and the direction always remain the same.
As the static version, this digital asset also has a link with the casting.otl and the external files of motion capture. It implies that every time that a new selection of animations is done it is required to fix those animations into the CHOP node moving the parameters of trim and cycle.

Depending on the obstacles’ shape and dimension the avoidance could produce a non-realistic outcome, because the agent adds a translation into their animation without raising his speed, so the avoidance path should have a prudent smooth shape.

Because saving variables for instants bigger than a single frame is a complex task, the logic to produce the avoidance between agents doesn’t work in some specific cases. Nevertheless, in a complete crowd simulation, this kind of errors won’t be visible and in a simulation with a low number of agents it works without errors.

Depending on the avoidance path of the obstacles, during the agents’ avoidance can occur a collision between agent and obstacle, because the agent is moving to avoid the other agent but he is not seeing the distance with the object, nevertheless the translation to avoid an agent is minimum and it represents a very unusual case.

7.4 Design

The crowd replication dynamic digital asset takes a big part of the static version to develop its function. As said in the description, the random assignment of the textures and the animations is kept, so the stamp variables are used again. The human_geometry, SHOP, Rigging and skeleton nodes remain the same, the CHOP is modified to have a 3D approach and to read more information about the motion capture files, and some nodes are created to calculate and to set the scene. This digital asset has been divided into 3 main stages: the setup stage, the interaction stage and the placement stage.

<table>
<thead>
<tr>
<th>Setup Stage</th>
<th>Interaction Stage</th>
<th>Placement Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skelet</td>
<td>move_v2</td>
<td>Rigging</td>
</tr>
<tr>
<td>Ground</td>
<td>CHOP</td>
<td>Skelet</td>
</tr>
<tr>
<td>obstacles and obstacles1</td>
<td>Ground</td>
<td>SHOP</td>
</tr>
<tr>
<td>move_v2</td>
<td>Obstacles and obstacles1</td>
<td>move_v2</td>
</tr>
<tr>
<td>human_geometry</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the setup stage the world is created and designed. The nodes involved in this stage have the function of create a specific component of the scene. These nodes are capable to import external geometries or to use the default geometries set. The nodes involved in this stage are: Ground, Obstacles, obstacles1, move_v2, human_geometry and the inputs from the casting.otl.

During the interaction stage, all the "brain" of the agents is designed to manage the information that is collected from the environment. The brain is centered in the move_v2 node, nevertheless all the information is pre processed in other nodes such as the CHOP node. The environment information is treated and returned to their original nodes so these nodes are part from this stage too.

And in the placement stage, the textures and the positions are set, basically is the stage where the static version of the asset is used. Plus special functions of other nodes like the move_v2 node that replaces the points_treatment and placement nodes from the static version.

The Houdini’s attributes management approach is highly used to have a complete communication between all the 3D objects within the scene especially in the interaction stage. Inside the digital asset there are 11 second level nodes in which each one have one or more stages' functions implemented.

During all the setup and interaction stages, the agents and the objects are not yet the geometries imported, but they are intelligent points that contain all its information. Just in the placement stage these points are replaced by the geometries.
7.4.1 Nodes

7.4.1.1 human_geometry
It remains the same node from the static version. Its function is to import the geometry of the human model and assign to it the skeleton, the movement and the texture, using the stamp variables from a copy node in the move_v2 node. For further information see CrowdReplication.otl, in the design chapter, human_geometry node. In this digital asset it makes part of the setup stage, and its function is to create and adjust all the visible parameters and attributes regarding the human 3D objects.

7.4.1.2 Rigging
Also remains the same node from the static version. Its function is to provide a modified copy of the skeleton formed in the motion capture files to make the rigging with the human_geometry. This node belongs to the placement stage. Its function is to attach the skeleton to the human created in the setup stage. For further information see CrowdReplication.otl, design chapter, Rigging.

7.4.1.3 SHOP
The materials used for the agents are the same. This node communicates the information of the materials to the node human_geometry to assign randomly the textures to the agents. This node also belongs to the placement stage, because here are loaded the external files for the textures that are applied to the human model loaded in the setup stage. For further information see CrowdReplication.otl, design chapter, SHOP.

7.4.1.4 Skelet
The library of Motion Capture used in this digital asset and the assignment method to the agents remains invariant from the static version. This skeleton uses the information loaded in the CHOP node to apply it to the agents. This node is part of the placement stage but also of the setup stage. Regarding the setup stage its function is to load the external files of the skeleton provided by the motion capture files. Regarding the placement stage its function is to assign the movement to the agents by communicating parameters with the other nodes. Receives the stamp parameter from the move_v2 and passes it to the human_geometry to deform the geometry imported there. For further information see CrowdReplication.otl, design chapter, Skelet.

7.4.1.5 Ground
This node as its name implies, is where the terrain of the animation in created. The importance of this node comes when it is needed to have this information to train the agents, namely the information of its composition needs to be sent to the agents, thus they can follow it without any problem. Inside the node is composed a predefined geometry that is done by deforming a grid with a mountain node, or can be used an external geometry specified in the parameters interface. Whichever of both is then subdivided and passed to the agents. The first thing passed to the agents (in the move_v2 node) is the geometry without scaling, and it will be used as the default placement for the agents. Then the geometry is scaled and is re-passed to the agents (in the move_v2 node also) and it represents the final ground of the animation that the agents will follow. Also the information created in this node is used to do the default placement of the obstacles. This node belongs to the setup stage because here the 3D
object world is created. Nevertheless this node is not visualized in the final scene; to do this task it's created the View_world node.

7.4.1.6 Obstacles and obstacles

These nodes are identical but independent, so the architecture is the same, but the outcome depends 100% of the user inputs in the parameter interface. Both nodes have the function of place objects in the scene. These nodes are divided in 3 subnets that have sequential functions. The first subnet regards to the creation of the points to place the obstacles. It is linked to a parameter in the parameter interface in which the user can select what to use. The default placement is the ground geometry scaled a little bit to avoid errors of placing obstacle too much close to the edge. The other option is to design in a new geometry the points where the obstacles will be using curves as described in the how to use section. The third option is to turn off the obstacle, namely don't use a placement and as consequence don't have objects of this type in the scene. Then these points are sent to the move_v2 node to be treated and accommodated into the scene.

The second subnet of this node is referred to the geometry of the object, also depends on the input of the user. The user can select from an LSystem geometry (thin obstacle), a cylinder (wide obstacle) or to load an external geometry of a custom obstacle. Then this geometry is copied to the points that return the move_v2 node.

The third subnet serves to receive the points from the move_v2 node and project it into the ground. For processing the points in the agent's brain it is necessary to lift the points to have all the 3D objects at the same level. Do this give the advantage to calculate all the distance in 2D, namely in the plane XZ because the Y component is the same for every point. When the points arrive back to the obstacle node they are lift yet and need to be projected into the terrain. Here is imported the final node of the ground node. The points are modified to have normals directed to the terrain. Then the points are projected using a ray node.

![Figure 34 Curve design to place obstacles](image)

Finally the points of the third subnet are used to give the position to the geometries of the second subnet. It is done using a copy node.

7.4.1.7 CHOP

The real design of the brain of the agent begins here. Contrary to the CHOP on move_v2, all the translation components of the root node are needed, nevertheless it’s better to use it separately and treat the each one in a different way. While in the static version the components X and Z were erased, in the dynamic version these components are used in order to get useful information for the agents, such as the speed of the animation, the rotation and the current position of the agent.

From the static version remains the same the way the files are imported, using the file master to have the list of the files and in the animations node is called this reference with the index read from the casting.otl inputs. Also remains the way the Y component is treated.

The explanation of how to fix the animations can be found in the CrowdReplication.otl section, there is explained how to use the nodes trim-cycle-shift to convert the animation into a cyclic long time and realistic animation. Regarding this digital asset the attention will be centered into the new features presented for the dynamic version of crowd replication.

The differences begin when the Z and X components need to be treated to have a realistic movements of the agents. Because the agents only have the human shape in the final part of the digital asset, the
points that will simulate the movement must have the information of the motion capture animation. To achieve this some information from the CHOP node is needed.

7.4.1.7.1 Rotation of the animation

The first thing to know about the motion capture files is the rotation. The Y component of the translation won’t give any information regarding the rotation. The first things that give information about the rotation are the motion capture files, watching the animations using the casting.otl, it is notable that the animations are only rotated 0, 90, 180 or 270 degrees, and it simplifies a lot the work. To identify the angle of rotation the solution implemented is to see the speed of the X and Z channels separately. In this order of ideas, to know the speed it is necessary to process the information of the file.

With just a single channel to manipulate (after apply a delete node with the name of the interested channel), it is necessary to resample the curve with the value of the frame per second, in this way the approximation for the speed will be better. The value of the speed that is needed is the average value for the frame range. After resample the channel it is copied into a null to have the reference. The next step is to transform the translation curve into a speed curve. There are 2 ways; the first is to use a slope node that applies the first derivative in time to the curve. The second way is to make an approximation of all the animation frame range; it’s to take the copied information from the null node and uses it to identify the speed. The approximation is to calculate the slope between the start and the end point, the expression is:

$$\frac{X_2 - X_1}{T_2 - T_1}$$

Written in Hscript:

```hscript
$FPS*(chopcf("../TZ",0,100)-chopcf("../TZ",0,-100.3))/(chope("../TZ")-chops("../TZ"))
```

Figure 35 Speed Calculation

Where: chopcf() evaluates the value of the channel in the first parameter at the frame in the third parameter.

chope() returns the value of the time of the ending point of the channel specified.

chops() returns the value of the time of the starting point of the channel specified.

Because the curves are used using frames not time the expression is multiplied by the FPS.

In this way the average speed is calculated for the two components. The speed calculated with the slope node serves as reference to see how far the approximation was. Knowing the speed of the components it is necessary to compare it and to package the information in order to can send it to the move_v2 node.

An auxiliary node is used for this scope. If the magnitude of the Z component is bigger than the magnitude of the X component it means that there is a rotation of 90 degrees, it can be clockwise or not. Into an auxiliary variable is stored the values 90 or 0. Then the result value of the speed of the components is compared also using the sign. If the Z component is bigger than the X component, it implies a rotation clockwise, so into another auxiliary variable is stored the value -1, in other case the stored value is 1. Then the auxiliary variables are multiplied to have the complete information of the
rotation in Z. For the rotation in X a final step is needed to be done. Because the multiplication of the variables in both cases is 0, the 180 degrees rotations are not taken into account. Another conditional is implemented, if the final value plus the second variable is -1, the rotation is 180 degrees, in other case is the value calculated in the previous variable. This sequence in Hscript is:

```
rota=if(abs(ch("../channel2/value0x"))>bs(ch("../channel1/value0x")),90,0)
r2=if(ch("../channel2/value0x")>ch("../channel1/value0x"),-1,1)
rt=ch("rota")*ch("r2")
rtt=if((ch("rt")+ch("r2"))==180,ch("rt"))
```

This variable called rtt is the output for the rotation information.

### 7.4.1.7.2 The current position of the animation

The other part of the CHOP information needed is the reference for the position of the animation. The movement of the points in the move_v2 node is ruled by the original animation, i.e. the points will move at the same speed, at the same frame of the animation. For this scope more information from the CHOP needs to be sent.

Once again the channels to be used are the X and Z channels from the translation node (Hips-root). Coherence between the translation and the animation position is necessary in order to have a realistic outcome of the animation. So the X and Z channels must to be related to the Y channel current frame. For this within each channel (X and Z) are used referenced copies of the nodes trim-cycle and shift used for the Y channel.

The information packaged to be sent to the move_v2 is the local value of each coordinate (x and z) and the initial value of the animation, i.e. after the shift and the offset applied in the animation, the value of the position in the frame 1. This is information will be used to convert this local values into global values for the animation, moving the animations from the first frame to the origin of the world. The information is packaged into a channel node that contains 2 values:

- The current frame value: `chopcf("../PSZ",0,0xFF)`, and
- the first frame value: `chopcf("../PSZ",0,1)`.

### 7.4.1.7.3 The animation

The output of the animations node is the information of the animation. The same method used in the static version is used to this scope. The difference is that the agents will move using the information previously described. The output animation is stopped in a fixed point, so if the animation is loaded in a wrong point, the outcome will be unrealistic because the animation will move with a different speed of the point’s one.

### 7.4.1.8 move_v2

This is the center of this digital asset. The main functions of this node are: to create the intelligent points that at the end will be replaced by the agents; to read and manage the information that comes from the other nodes (CHOP, Ground, obstacles and obstacles1); to set up the first frame to avoid errors; to place the agents into the scene; to create the visual guidance for the help; to manage the interaction between the scene’s 3D objects.

Using an Artificial Intelligence and a neural networks paradigm, this node receives the information from the environment to produce the most appropriated response of the agents. The treatment of the information has been divided in several subnets sections that have also divided functions. These subnets follow a sequence, but some tasks are done in parallel.

Since the information is read and packaged in the CHOP, but it is not manipulated, in the move_v2 this information is received and stored as attributes that describe each point. The first subnet is called **Points_Creation**. An initial base point is created and a list of empty attributes is also created for it. These attributes will be filled with the information that arrives from the CHOP node (the speed in the x component, the speed in the z component and the angle of rotation).
Then other attributes are created to characterize the points. In the copy node the necessary number of points is created (set the parameter number of copies that is linked to the number of agents’ parameter in the parameter interface) and using the stamp variables the attributes created before are filled with univocal values for the points:

For the attributes that receives the information from the CHOP node is used the following expression:

```ch("..//CHOP/animations_"+stamp("..//copy1","i",0)+"/CHANNELNAME/VALUENAME")```

This expression reads the information stored in the specified parameter. The stamp function will return the number of the meaningful name for the animations_NUMBER node, so the information received will be inherit from the animation loaded there. For the three parameters is the same expression, just change the channel’s name and the value’s name.

Then the two values of speed are reduced to a single parameter by selecting the right parameter based on the angle of rotation:

```if($ROT==0 || $ROT==180 ,abs($SPEED1),abs($SPEED2))```

The other attributes created represents the univocal values for each point, namely are created the animation index and the point index attributes. The first one is the reference to the animation loaded and will be used after to set the position of the point while the second parameter is the id of the point, every point will have a different number.

After the copy node is received the information of the initial offset of the points. This initial offset is called agent’s placement in the parameter interface. To match with the number of points created, the placement is specified with a primitive and a scatter which number of points is also linked to the value of copies of the copy node node is used to create the points then. The information of the position in X and Z is stored in a vector attribute called oi; attribute that will be copied into the points. With all this information collected is time to set the first information to the recently created points. With a point node the position of the points is set. Because the plane of translation is the XZ, the information of the Y channel is ignored (by now).

To set the position in X and Z, is necessary to pick some rules and to know the information available. The main restriction is that the agents will always follow a linear path, so the X position is also ignored (by now), the difference of the translation in this axis is smaller than the translation in the Z axis. Then the X component will be used to avoid collisions. The information to take into account is the rotation of the animation, the oi attribute, the current position and the initial position (from SHOP). It could be easy to select the direction of the animation by doing a switch and selects between 4 type of points, one rotated 90 degrees, other 180, other 270 and the last one without rotation. In fact this was the first solution, nevertheless it’s better to use a trigonometric expression to set the position of the points.

The main idea is to put all the animations in the same direction. Depending of the degrees of rotation the value in X or Z must take the information of a channel of the CHOP. The conversions to do are:

In order to make the current position dependent from the angle and the positions read in the CHOP, a similar procedure to the conversion to polar coordinates is done. The value of the axis will be the combination of the magnitude of the movement in the axis multiplied by a trigonometric function.

```X= posX*cos(rotation)+posZ*sin(rotation)
Z=-posX*sin(rotation)+posZ*cos(rotation)```

In the case of the angles of rotation, always one component will be deleted, so the conversions of the table are done. Nevertheless, for the points there are other parameters that must be taken into account and these parameters will modify the magnitude of the movement. The posX and posZ described in the equations above are set by the current value of the animation minus the initial value of the
animations (they are local values). The initial offset from the oi attribute must be added in the respective coordinate. So the expressions for the positions written in Hscript are:

For X:

$$O_{11}$$

For Z:

$$(O_{12} + \left\{ -ch("/../CHOP/animations_{"+IDX+/ch_x_out/value0x") +ch("/../CHOP/animations_{"+IDX+/ch_z_out/p0") \right\} *\sin(ch("/../CHOP/animations_{"+IDX+/mmm/ rtt") + \left\{ (ch("/../CHOP/animations_{"+IDX+/ch_z_out/value0x") - ch("/../CHOP/animations_{"+IDX+/ch_z_out/p0") \right\} *\cos(ch("/../CHOP/animations_{"+IDX+/mmm/ rtt").

As said before, the movement in the X axis is ignored because the magnitude of the movements is depreciable. While the Z axis the movement will follow the animation higher speed direction, translated to the origin and moved to the initial offset set by the placement input.

Then the setup of the scene’s first frame is done. Two subnets in parallel do this work. They are called Delete_points and Delete_Initial_Obstacles.

In the Delete_points subnet the information of the first frame is stored. All the points must be compared with the other to evaluate the proximity. A particular paradigm of programming must be applied here. In a common language it’s enough to put a cycle and evaluates the condition for all the points, nevertheless this implementation increases the calculus complexity and Houdini provides another option to do this task.

The geometry with all the points enters to the subnet. To have just one point in to compare is used a delete node which will delete following a pattern. This pattern is a stamp function from other copy node that is at the bottom of the subnet. Another delete is used to exclude the point and can compare and this delete is ruled by a filter in which the points that have a lower index are not taken into account, because this comparison has been made when these points were in the single point geometry, so the input is divided in two different geometries, one with one point, the other with the number-1 points. Then in both sides an attribute called col is created. For the single point its value is 0, for the other points the value is 1. Using an attribute transfer node in which depending of the distance (controlled with a threshold) the attributes set are passed from the right to the left. So if the single point collides with another point its attribute col will change its value to 1. Then a copy node is placed to make the cycle and compare all the points. This procedure is done in a copy of the output of the previous subnet. To warranty that the information is taken just for the first frame is used a timeshift node that permits to fix the reference frame.

Then to copy the information of the col attribute into the original points is used an attri copy node. This node copy from the right to the left the attributes listed. After that is needed to take the action, and it is to delete all the points that have the attribute col equal to 1.

Here a correction to the system must be done, because each point has a number point assigned by Houdini and during the deleting this values change. This thing affects the animation in the way that the relation between point and animation is lost. To fix it before was created the attribute AAA that is the univocal index for the points. A sort node permits to order the points by parameter. Naturally the points are ordered with the AAA attribute. The output of this sort node will represent the current “original points”.

Parallel to the Delete_points subnet is working the Delete_Initial_Obstacles subnet. The function of this subnet is to avoid the problems that can present the obstacles between them or with the agents on the first frame.

Once again is used a copy of the position of the nodes in the first frame. Also it is needed that the information of the placement of the obstacles is read at this part with some nodes of type object_merge. It’s important to have the points at the same height, so before comparing, in all the sides is put a point node to fix the Y component of the points. Using again the attrib transfer node, but
in a different way, the points are compared. These time at both sides there are the entire number of points, on the left side the obstacles, on the right side the agents. The node compares all the points from one side to all the points of the other side. The parameter that is transferred is called rmo (from remove obstacle) and in a similar way of the delete_points subnet, the obstacles (points) that has this value equal to 1 are removed from the scene. Because the comparison is done with the agents and not with the other type of obstacles a third comparison must to be done, because can occur that one type A obstacle is placed in the same position of a type B obstacle. The same procedure is done, and the parameter used is called rmoo (from remove obstacle from obstacle). The obstacles deleted at this point will be always of the type B. All the thresholds to calculate the proximity are controlled by the user in the parameter interface; these values are linked to these nodes.

The outputs of these parallel subnets are:

Points of the Agents, Points of the Obstacles Type A, Points of the Obstacles Type B.

These outputs will represent the inputs of the next subnet: **Obstacles_Avoidance**.

It is important to understand that the points of the agents vary their position on time, while the obstacles are static during the animation. The subnets above were just looking the proximity in one fixed frame. This subnet calculates every frame the proximity between the dynamic agents with the static obstacle, so it can notice a change in the state of the agents. Basically when an agent gets into the proximity of an obstacle changes his state to divert his way from the linear path. This divert is modeled by a parameterized ellipse which radius can be manipulated from the parameter interface.

The paradigm to control the distance between the agents and the obstacles is the same than the used for delete the initial obstacles. But this time they aren’t erased, this comparison is used to define the behavior of the agent in front of the obstacle. In order to identify the possible collision are created another two parameters in which will be stored the id of the type of obstacle and the coordinates X and Z of its position. These parameters are created for the agents and the obstacles. Each type of obstacle is handled separately and then is compared in cascade with the agents. For the obstacles are added two important things; the first one is the manipulation of the points normals to make the projection, after the change of the normals the points are packaged into a null to be sent back to the obstacles and obstacles1 nodes. The second one is the implementation of the visual guidance, these points positions are used to design circles around them. Once again an attribute transfer is used to give the information to the agents, and after that is created the avoidance parameters.

![Figure 36 Objects Avoidance](image)

The avoidance of the obstacles is modeled with a parameterized ellipse. The idea is to divert the path in the x-axis, for this reason some attributes to control this ellipse are created. The first two parameters are used to control the radius of the ellipses, these parameters called “rad” and “radz” are vectors of two positions (because there are 2 types of obstacles), an implementation for more type of obstacles is possible, but it requires to do some changes in this node and create another obstacles node. The third parameter is called “relpos” (from relative position) and basically is a vector of 3 positions parameter that flags some changes. In the first position is stored one if the agent is in the collision zone on the right side, a 0 if the agent is in the collision zone on the left side and -1 if the agent is outside the collision zone. The second position has the same function but using the z-axis, so there will be one if is behind the object, 0 if is in front of the object and -1 if there is no collision risk. The third position is used as reference for the translation in the x-axis.
At this point another help for the visual guidance is implemented. Depending on the values of the first position in the relpos attribute, the agents are colored to identify the action that they are executing. If he is taking the left side of the ellipse is colored blue, if he is taking the right side is colored green and if he is following the normal path is colored white. Then based on the parameters created with the information, are created other attributes to process the information. They are called refe (from reference) and alpha. Refe only watch if the agent is taking an evasion action and puts a number 1 in this parameter otherwise puts a 0. Alpha represents the variable translation of the agent within the x-axis. Alpha is the difference value between the actual position of the agent and the position of the ellipse border.

In order to create the deviation of the agents some values are required. The first values are the coordinates of the center of the obstacle, that are stored in the attribute OPOS. Other values needed are the actual position of the agent, stored in the global variables $TX and $TZ. Finally a user input is required to control the dimension of the ellipses, these inputs are stored in RAD and RADZ.

And using the canonical form of the ellipse is possible to deduce the alpha value:

\[
\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1
\]

Where:

\[
X = TX - OPOS1 + ALPHA; \quad Y = OPOS2 - TZ; \quad a = RAD; \quad b = RADZ
\]

Replacing these values into the canonical form of the ellipse and clearing alpha the value of alpha is:

\[
ALPHA = RAD \times \sqrt{1 - \frac{(OPOS2 - TZ)^2}{RADZ^2} - \frac{(OPOS1 - TX)}{RAD^2}}
\]

And in Hscript this expression turns into:

\[
RAD1*sqrt(1-((OPOS2-TZ)*(OPOS2-TZ)/(RADZ1*RADZ1)))-abs(OPOS1-TX)
\]

The value of alpha depends on the obstacle type, so the values of RAD and RADZ are conditioned to the obstacle id stored in the attribute OBS.

With the information processed it’s time to modify the path of the agents, by now the information is processed and calculated but not yet applied to modify the behavior of the agents.

<table>
<thead>
<tr>
<th>alpha</th>
<th>refe</th>
<th>relpos1</th>
<th>OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0</td>
<td>1</td>
<td>-1</td>
<td>CONTINUE</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>GO LEFT (Add Alpha)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>GO RIGHT (deduct alpha)</td>
<td></td>
</tr>
</tbody>
</table>

The expression to change the agent’s path is reduced to the table below. First it is necessary to verify if the alpha value is higher than 0 and if the refe attribute is 1. Then with the information of the attribute RELPOS1 (the horizontal relative position) is applied the increment alpha to the actual position of the agent. The expression in Hscript is applied to the X component of the translation in a new point node that will modify the actual points:

```hscript
if($ALFA>0,
if($REFE==1,
if($RELPOS1==1,$TX,
   if($RELPOS1==1,
      $TX-$ALFA,
      $TX+$ALFA
   ),$TX),$TX)
```

The Y and Z variables maintain their precedent value ($TY and $TZ). And making this the obstacles avoidance is completed. The step to take at this point is to prepare the output points of this subnet to
make a mutual avoidance. Before the subnet of avoidance, two subnets for the creation of the attributes are placed.

The first subnet is called **Attributes_creation_for_avoidance**. On this subnet is considered the information of the actual frame and the information of the next frame. This subnet is divided in two parts in which each one begins with a timeshift node. These timeshift nodes fix the current frame and the next frame in each part of the subnet. In both cases the main scope is to see if the agents are close enough to say that they are risking a collision. Once again using the attribtransfer these in information is passed and stored in the attributes fc and fca (each one in one part of the subnet). In the case of the current frame is using (at the same time of the fca attribute) an auxiliary vector attribute to pass some information that will be useful to the design of the evasion action. This information is all the attributes that describe the possible collision target, such as the speed, the current position and the point index.

All these attributes have been stored previously with the names: $SPEED$, $AAA$, $TX$ and $TZ$. The procedure for the attribtransfer is the same procedure described in the delete_points subnet; each point is compared with the others and then with a copy node the cycle is set. In this case, the copy node rules two cycles at the same time (the current frame and the next frame). The union of this nodes is done with an attribute copy node, in which all the parameters created (fc, fca and aux[4]) are merged into the same group of points.

The next subnet uses the information passed from the previous subnet and put it into the actual setup of the points (the output of the obstacles avoidance). The scope of this subnet is to calculate the Euclidean distance between the points that have been found close and it is called **Distance_between_collision_points**.

It uses the actual position of the collision target and the index of the point to determine the distance. If the index of the collision target is the same index of the points that is being analyzed the distances is set to 100 that is a value that means a huge distance for not being taking into account as a possible collision. If the index of the collision target is different, the value of the AUX positions are different to the values stored in TX and TZ, so the distance is calculated as the magnitude of the vector that starts at ($TX$, $TZ$) and finishes at ($AUX2$, $AUX3$). The Hscript expression for the distance is:

$$\sqrt{(TX-AUX2)^2 + (TZ-AUX3)^2}$$

After calculate the distance the nodes must be re-ordered using a sort by $AAA$ (point index) node. With all this attributes created and ready to use the design of the evasion can be implemented. The first thing to do is to copy the created attributes to the current point group with an attribcopy node.

The subnet where the evasion is designed is called **Avoidance**. Here the attributes created above are processed and a logic schema is constructed to select the agents’ decision. First of all a visual guidance is created, the agents that are risking colliding turn their color to red, in this way to identify possible errors during the animation will be easier. Then in order to make the information received compatible with the logic the parameters are transformed into 3 flags attributes that will rule the logic. The first flag serves to identify who is slower, the agent of the collision target, the flag is 1 if the agent is slower, otherwise is 0. The second flag sees which one is on the right, if the agent is, and then is stored a 1, if not then 0. And the third one indicates if the slower one is behind; uses the information of the first flag and the respective information of the Z axis to compare.

In an analog way to the obstacles avoidance, the evasion between agents is modeled as a translation in the X axis. The difference is that here both parts are dynamic, so both parts must execute an action in order to avoid the collision. The evasion is a circular path calculated from the position of the agents. If the agent is on the right it will move following the right arm of the circle, and if the agent is on the left it will move following the left arm of the circle. The translation radio is set by the user in the parameter interface. Similar to the alpha value on the obstacles avoidance a coefficient to be added is calculated and is called beta.
Beta is calculated using a parameterized circle. The projection from one point to the edge of the circle represents the translation, nevertheless here both elements are moving so the movement must be soft because the circle changes in every frame. Also the speed of the agents must be taken into account because it will be related to the duration time of the evasion. Projecting both agents into the contrary circle will produce a repulsion effect, the distance between them will always be the radius set by the user, nevertheless if the distance to move is the same to both agents, it will produce that only one agent will move and it could produce a non-realistic outcome. So the calculated value will be manipulated in when the same is applied to the agents.

![Figure 37 Agents Avoidance](image)

The logic is based on the attributes created previously. A deeper reasoning tells that in a possible collision the faster agent must be behind, otherwise the distance will increase. Another parameter that modifies the behavior of the agents is if the agents are avoiding an obstacle. This fact is measured using the attribute relpos[0], where the information of the avoidance of the obstacle is stored and codified with the values: -1 if there is no avoidance, 0 if the avoidance is to the left and 1 if the avoidance is to the right. If the avoidance of an obstacle is being executed, the values for the agents’ avoidance must change in order to produce a re-collision with the obstacle. In this order of ideas, the translation for the one who goes in the inner part of the curve is reduced by 1/3, and the translation for the one who goes in the outer part will remain beta. Also to know the horizontal position gives the possibility to decide the direction of the movement, so the decision will be defined like this:

<table>
<thead>
<tr>
<th>dist</th>
<th>flag</th>
<th>flagc</th>
<th>relpos[0]</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;10</td>
<td>0</td>
<td>1</td>
<td>0 or -1</td>
<td>Go Left ($BETA/3)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Go Left ($BETA/3)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Go Right ($BETA/3)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0 or -1</td>
<td>Go Right ($BETA/3)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0 or -1</td>
<td>Go Left ($BETA/2/3)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Go Left ($BETA/2/3)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Go Right ($BETA/3)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0 or -1</td>
<td>Go Right ($BETA/3)</td>
<td></td>
</tr>
</tbody>
</table>

And this logic translated into the Hscript language to modify the behavior of the agent is:

```hscript
if($DIST > 15, $TX,
   if($FLAG==1,
      if($FLAGC==1,
         if($RELPOS1==0 || $RELPOS1==-1, $TX+
            $BETA*2/3, $TX+$BETA/3)
      ,
         if($RELPOS1==0 || $RELPOS1==-1, $TX-$BETA/3, $TX-$BETA*2/3)
      ),
      if($RELPOS1==0 || $RELPOS1==-1, $TX+$BETA, $TX+$BETA/3)
   ),
   if($FLAGC==1,
      if($RELPOS1==0 || $RELPOS1==-1, $TX+$BETA, $TX+$BETA/3)
   )
)
```

63
Using this logic, the faster agent must follow a bigger path and the slower one a smaller path. A special difficulty to develop the logic is the fact that Houdini can’t save variables from one frame to another, reducing the possibilities a lot. It means that the interaction must be calculated in the current frame, losing some important possibilities to change the agents’ behavior. Nevertheless the outcome achieved with this implementation is really good because the agents will produce a reaction to the proximity and it is near-realistic.

After the agents avoidance it’s time to place the points in the world, namely the movement has been modeled but is lacking some important things like the agents rotation (because they are always watching to the Z axis and they are making unreal curves in the avoidance) or the manipulation of the Y axis (the agents are lift and all are in the same height).

The following subnet uses the information of the current frame to compare it with the information of the fourth previous frame. The selection of the fourth previous frame is to smooth the difference of the angle of rotation, because taking the previous or the second previous, when the agent is closer to the end of an evasion movement the angle of rotation changes a lot. The information of the X and Z coordinates of the agent 4 frames before the current is stored in an auxiliary attribute called t. Then these attribute is copied to the current point group (the output of the avoidance subnet) and the rotation can be calculated. This subnet is called Anim_rotation.

With this situation, the angle of the rotation is the arctangent of the triangle described above. This angle is calculated and stored in a new attribute called AROT. The value of this attribute in Hscript is:

\[
\text{atan}((\$TX-\$T1)/(\$TZ-\$T2))
\]

Finally the value of the angle is copied to the current points group with an attributecopy. The last modification needed before placing the agents in the scene is to make the points follow the terrain and this is the function of the next subnet. The follow_terrain subnet imports a copy of the final terrain, and using a ray node the points are projected on the terrain. After that a correction to the normals must be done. Because if the normals remain with the negative value in Y, the agents won’t be perpendicular to the ground, will be parallel. Using a point node, the normals are corrected changing the value of the X and Y normals to 0 and the Z normal to 1. Doing this, the manipulation of the points is completed and the agents are ready to be placed in the scene.

The last subnet is called geo_copy and its function is to import the human geometry, place a transform node to modify the geometry dimension from the parameter interface and copy this geometry to the points. This copy node uses some stamp variables to make a random assignment of textures and animations to the agents. Other stamp variables are used to copy the information of the color of the agents for the visual guidance help. The assignment of the textures and the animations is done in the same way of the static version. To see the complete explanation of how it is done, see the CrowdReplication.otl section. The output of this subnet is packaged into a null and this null will be displayed on the final scene.
7.4.1.9  Help
This is a visualization node. In this node are grouped all the visual helps that were designed in order to activate or deactivate all at the same time. Within these helps are the ellipse for the obstacles’ avoidance and the placement zone. When it’s active it corresponds to a Help View mode.

7.4.1.10  View_world
Also this is a visualization node. In order to select the 3D objects that will be displayed on the final scene, this node imports the output geometry from the ground, obstacles and obstacles1 and with some switch nodes that can be handled from the parameter interface merges the selection to display it. Depending on the selection, the objects displayed corresponds to the Entire-World view and to the Non-visible Obstacles view mode.

7.5  Performance Datasheet
In order to evaluate the performance of the digital asset within the virtual environment, Houdini provides a tool called “Performance Monitor” in which is displayed selected items to see. All the tests have been done in the same computer, so the comparisons done correspond to tests executed with the same conditions.

To evaluate the digital asset some information is taken regarding the frame time and the time that is taken to make the copies on the node move_v2. The test scenario was the same for all the probes:

**Number of Obstacles of the first type:** 20
**Number of Obstacles of the second type:** 2

The same ground geometry.

**Variables:** Polygon Percentage; Visualization Mode; Display Mode.

**Case 1:** Polygons: 4%; Smooth Shaded; Entire View
**Case 2:** Polygons: 4%; Wireframe; Entire View
**Case 3:** Polygons: 10%; Smooth Shaded; Entire View
**Case 4:** Polygons: 10%; Wireframe; Entire View
**Case 5:** Polygons 4%; Smooth Shaded; Just Agents
**Case 6:** Polygons 4%; Smooth Shaded; Non-Obstacles View
**Case 7:** Polylons 4%; Wireframe; Help View

<table>
<thead>
<tr>
<th></th>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
<th>Case 4</th>
<th>Case 5</th>
<th>Case 6</th>
<th>Case 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>FT</td>
<td>CT</td>
<td>FT</td>
<td>CT</td>
<td>FT</td>
<td>CT</td>
<td>FT</td>
</tr>
<tr>
<td>1</td>
<td>663,62</td>
<td>1,72</td>
<td>639,57</td>
<td>1,82</td>
<td>664,98</td>
<td>1,88</td>
<td>637,51</td>
</tr>
<tr>
<td>10</td>
<td>1770</td>
<td>8,93</td>
<td>1750</td>
<td>6,94</td>
<td>1760</td>
<td>11,15</td>
<td>1730</td>
</tr>
<tr>
<td>20</td>
<td>1840</td>
<td>18,25</td>
<td>1840</td>
<td>12,93</td>
<td>1810</td>
<td>26,68</td>
<td>1790</td>
</tr>
<tr>
<td>50</td>
<td>2130</td>
<td>68,62</td>
<td>2010</td>
<td>48,61</td>
<td>2120</td>
<td>131,1</td>
<td>2160</td>
</tr>
<tr>
<td>100</td>
<td>2730</td>
<td>228,27</td>
<td>2480</td>
<td>202,47</td>
<td>2890</td>
<td>446,03</td>
<td>2930</td>
</tr>
<tr>
<td>200</td>
<td>3580</td>
<td>533,88</td>
<td>3690</td>
<td>683,88</td>
<td>4820</td>
<td>1480</td>
<td>4660</td>
</tr>
</tbody>
</table>

A: Number of Agents; FT: Frame Time (ms); CT: Copy Node Time (ms)

The table above shows the times that the computer takes to calculate and display the whole frame (FT) and the time that takes the copy node on the move_v2 subnet (CT), for the various cases proposed before.
The frame time is independent from the number of textures used; it will represent just a little delay smaller than a millisecond. But when there are several animations selected the process to read the .bclip files increase the time of rendering. All the tests were done with 4 animations selected, nevertheless when this number was increased in other tests, the time of calculation increases. The reason is that all the animations are referred to external files, and when the number of animations selected rise, a lecture of the memory is required, depending on the type of storage hardware the delay will be greater or minor.

From the experimental values it is seen that the type of visualization doesn't affect too much the time of calculation. Nevertheless the response was the expected according to the theory: the tests done using wireframe visualization takes less time to construct a frame. When the number of polygons is increased (cases 3 and 4), the same occurs with the frame time when are used more than 50 agents, the delay is increased and the copy node takes more time to reproduce the scene. However, the difference between the times also depends on the machine type in which the digital asset is used and in the experimental values it's possible to see that the smooth shaded tests takes more or less the same time and depending on the memory available, sometimes can be lower.

The increase of the delay when more agents are used is because the use of the ram memory; the more agents are used the more difficult will be allocate the data on the ram memory. For this reason the first copies are done much faster than the latter copies. A solution for this problem could be cache the data before rendering.

7.6 Problem Solving

7.6.1 Caching the Animation

In order to introduce the possibility to watch the scene in a fluid way before doing a render, it is used a calculation of the geometry to reduce the calculus for the final render. Houdini provides an important tool called ROP output driver that works only in geometry level and serves to save the animation of the geometry into a .bgeo file for a set frame range.

For the Dynamic version this tool is applied for the output node of the move_v2 subnet and for the output of the View_world subnet. In the case of the View_world, it is necessary only to calculate one frame, because it is a static environment in which the objects would not suffer any deformation during the frame range. While for the move_v2 subnet it is necessary to indicate the frame range to evaluate. For the static version the tool is applied to the placement output node, and also a frame range must be specified.
The main vantage is the achievement of a fluid visualization for the crowd. With the geometry calculated previously, the movement of the agents is not calculated anymore. Using this tool error detection can be done easily, because it permits to visualize the scene within the 3D viewport so can be changed the point of view, not only the camera view position as the flipbook does.

A second advantage is the possibility to make an instance copy from the calculated geometry. Using this .bgeo files, and loading it into a file node, can be used the instance copying by making this geometry instance of other geometry. It will repeat the same pattern but using at least 100-men geometry it wouldn’t be notorious.
8 Future and Conclusions
The scope of this project was to create cheap and easy tools to produce different crowd replications. The implementation of some interesting features within Houdini Escape permits the user to create the simulations with a cheap commercial license, and for most cases avoid spending a great amount of money in specialized programs such as Massive.

The advertising market needs fast, easy and high quality solutions to show the merchandise, over time requires more and more shocking features to capture the public attention. These tools represent an enormous basis to the development of more and more complex crowd simulations. One of the principal objectives was to create a reusable tool that help the user to save time and money, however, besides of being reusable these tools have the advantage of also being modifiable from their root as the requirements of every single animation. Obviously to achieve a functioning change into the digital assets, it's required a big competence using Houdini and its environment. Another important feature is that all the default animations, geometries, textures, etc, are free to use, however for specific projects, the tools allow changing the animations, geometries and texture in an easy way. This vantage permits to build or buy different setups for achieving a unique outcome.

Nowadays the use of extras in scene is tending to be avoided because those people are always on a second plane where is not needed a great resolution and as a consequence can be replaced easily by digital human-like copies. The correct use of these tools permits the fast placement of these agents on a 3D environment.

The performance of the tools is very acceptable in order of the quality of the result versus the frame time. For the static version a huge amount of agents (over 5000) can be used without taking more than 2 minutes by frame. While for the dynamic version these time increase because the Artificial Intelligence algorithms are calculated every frame producing the ideal result but causing a delay on the frame time. Regarding the execution, the dynamic version provides a reliable performance. The AI algorithms that have been implemented produces a very accurate response for the agents, however, some problems couldn't be solved in a 100% efficient way and in some specific cases like 3 or more agents proximity can produce a undesirable outcome. The static version provides a simple but at the same time versatile tool. Simple because the it's use and parameter interface is very reduced and simple to manage. Versatile because permits a more extensive manipulation of the animations, producing a notable variety of results.

The tools are inserted into an actual market. Being created in a professional environment, the tools found an immediate application producing optimal results. Just 15 days after the creation of the static first version, it was used to fill the stadium of the TIM falling star spot. In the future, these tools can provide an excellent option for the company to fill the requirements where multiplication of agents (not only human beings) is needed.

Nevertheless, these tools represent a basic approximation to the crowds simulation because there are much more features that can be implemented but require much more time and knowledge to be done. Between the features to be done can be found the interaction between different crowds, the automatic LOD control, the infinite change of animations for each agent, for the dynamic case the possibility to change the path, and many others features that can be added as needed. Every day arises a new problem that needs an answer, for the case of crowd replications those problems are specific behaviors that the agents will assume during the animation. Taking the example of INVICTUS, more than 300 digital assets have been created by CIS Vancouver to have the crowd control of the stadium scene; every digital asset was used to produce a different reaction for the agents. With the appropriate manipulation of the digital assets created on this project, some new digital assets can be created with specific function such as trigger a wave in a stadium or to excite the people in a simulated concert.

Where to go?
Nowadays the crowd simulations are taking force in the visual effects field. On TV, advertising or Cinema can be found more and more backgrounds of stadiums, concerts, streets, manifestations, wars, marathons, etc. that really don't use real people and produce a big impact into the spectators. Another
aspect that affects this kind of effects is the increase of the computer capability to calculate, if the animation uses more polygons, more realistic it will look like, but also more complex to be calculated will be. Here begins one dilemma: increase resolution of the agents or add more features to the agent's "brain" or both?. In a not far away future both can be produced, nevertheless actually the entertainment industry is getting transferred from specialized offices and studios to the personal computers, changing the belief of that things can be only created by few ones to believe that every person can produce their own features. This thing implies the born of more and more complex technologies for entertainment, and the things created on the personal computers would not satisfy the requirements of the actual market, making that the surviving studios increase also their capabilities and their product quality. The tools developed on this project represent a basis where complex features can be built up.

They are constructed to supply the requirements described on the paragraph above; they can increase its resolution, but also are modifiable to reach new capabilities and extend their functionalities. As it is said previously, there is much where to build from these tools, many different cases and approaches can be branched from these tools. Extend to animals, produce a required effect in the right moment or improve the human qualities by changing textures can be listed as possible future additions for the tools.
9 Bibliography

Appendix A - Software Description

Houdini

General

Houdini is a high-end 3D animation package developed by Side Effects Software which headquartered in Toronto, Canada. Its chief distinction from other packages is that it has been designed as a purely procedural environment. A version of the product, called Houdini Apprentice, is available as a free download for non-commercial use.

Actually this software is used on several productions, such as "The Curious Case of Benjamin Button", "Transformers", "The golden compass", "Spider-man", "Avatar", "Invictus" and many other blockbusters.

Side Effects presents two different versions of the program: Houdini Escape and Houdini Master. The first one offer a cheap version (its license cost $1995 USD) where almost all the functionalities are presented except the dynamics' support and the particles' support. The second is the complete version (its license cost $6995 USD). The Houdini apprentice is Houdini master with limitations for rendering resolution and puts a watermark in all the rendered images, also there is an apprentice license that cost $99USD and permits to increase the max resolution to 1920x1080 and removes the watermark: this version is focused to create portfolios and develop personal projects.

The functionalities presented are:

- Nodes and Assets, Modeling, Animation, Character, Lighting, Rendering, Compositing, Scripting, Particles*, Dynamics*

*Only Houdini Master

User Interface

The Houdini interface is highly configurable; allow multiple views into different contexts of the package. These views are called Panes. There are multiple types of panes to view each of the Contexts in Houdini.

Viewer Pane

On the viewer pane there are the 3D viewport on the center, the selection toolbox on the left, the selected object parameters on the top and the display toolbox on the right.

This pane allows the user to see the 3D scene in real time by displaying all the 3D objects and their respective transformations on the screen. The 3D viewport shows also floating reference axis to indicate the user how goes the 3D world that can be deactivated with the toolbar options on the right side of the viewport or by pressing the "d" key and selecting the checkbox that will appear on the pop-up window. Also on the viewport can be specified the view (perspective, top, front, right, etc).

This viewport can be navigated in an easy way using the mouse and the spacebar of the keyboard: to zoom in or zoom out press the spacebar and the RMB and move the mouse, to rotate press the spacebar and the LMB, to move the reference press the spacebar and the MMB, to return to the initial view press spacebar + H and to have the view of all the objects on the scene press spacebar + G. Also the 3D objects can be selected by clicking and to dive from the object level to the geometry level is enough to double-clicking. On the geometry level the selection toolbox plays an important role.

The selection toolbox presents a list of options in order to select objects and geometries within the scene. The first option permits to modify the target 3D objects, namely to enable the selection of geometries, cameras, lights and other 3D objects that can appear on the scene. The second option is the geometry selection and serves to change the selection mode to points, vertex, primitives or edges.
Other two selection modes allow selecting the particles or the dynamics on the scene. Then the second part of this toolbox allows modifying objects by manipulating handles that appear also on the viewport. Those options are select, move, rotate, scale and show handle (where are all included). Below these options there are the snapping options and the view options. When the view option is enabled is the same thing to have the spacebar pressed. And the last option is the render region, where a part of the viewport is selected and a pseudo-window displays how it is seen after rendering.

The display toolbox gives some important tools to manage visualization helps. The upper options are referred to the scene context, those options permits to activate/deactivate the construction plane, to lock cameras/lights to the navigation of the viewport, use headlights instead of the object lights and to show the shadows. On the middle of the toolbox the options are the guidance while modeling, namely the options to activate/deactivate the visualization of the points, the primitives, vertexes and their respective numbers, normals, names, etc. and on the lower part of the toolbox there are some other options to visualize guidance items, like background images, custom attributes, etc. And finally there is a button to open the pop-up window of the display options (as said before its shortcut is to press the d key).

The parameters toolbox appears on the top when an object is selected and there will be the main parameters to modify it. These parameters vary according to the selected object. Above this section also there are the options to change the visualization mode (wireframe, bounding box, smooth shaded) and to show and/or create some objects on the current context

**Instant Objects**

On the upper part of the screen there is a shelf with list of objects that can be placed on the scene by just clicking them. Also there are options to modify them then like on the tab polygon. the polyextrude or other deformation options. The second part of this shelf is different from the two versions of Houdini because there are the options to set the lights and cameras but also the options to set the dynamics and the particles.

**Network Pane**

A Network Editor pane is at the core of Houdini's functionality. All operations in Houdini use nodes of various types to achieve a unified, logical, procedural interface. These nodes are called Operators, and are shortened to the operator type.

This node-based approach makes learning Houdini very easy, once you grasp the basic concept of wiring nodes together, since everything in Houdini (particles, channel editing, compositing, modeling, shader creation etc) is ultimately done with appropriate Operators.

**9.1.1.1 Parameters Pane**

On the upper right side of the screen there is another pane and there is the entire list of parameter to modify the selected object/geometry. Here the parameters can be changed by varying the values or by writing an expression that can be written on Python or Hscript. Also animations can be set by key framing the parameters.

To set a key frame it's necessary to press the Atl key + LMB, to remove a key frame Ctrl + RMB. Also the parameters can be displayed as expression or as string. As string means that the value that appears is the value of the parameter on the current frame. As expression means that it is the script that will rule the behavior of the parameter during the entire animation.

Beyond the panes, Houdini provides also other pop-up windows and pane options in order to monitor attributes, movement’s curves and performance; and to write external scripts on Hscript and Python. During the project most of these utilities are used to control the animation channels, the attributes transfer and the performance of the digital assets.

**Other Windows**
Details View: It serves to control the attributes’ values of the geometry of the selected node on the current frame. The attributes can be set on vertex, point, primitive and detail level, and the details view provides the option to select the view for the concerning context. During the digital assets is used to control the transfer of point attributes between agents, obstacles and world.

Motion Viewer: It is used to control the movement channels of a geometry. This window will only display something when a chop network is selected. It serves to control the cinematic curves of the translation and rotation of each point/primitive/geometry that is passed to the chop as reference. It only display, the curves can be manipulated by nodes at CHOP level or using the Channel Editor. This motion viewer is used while the animations are being fixed in order to control the TRIM-CYCLE-SHIFT sequence.

Channel Editor: Basically it serves to manipulate and change the channel curves by the translation of the points on the key frames. There can also be changed the interpolation functions to make the animation more realistic. For animations loaded from motion capture files it is not that useful.

Performance Monitor: This window permit to control the performance of the computer while executing one change (understood as frame change, load a file or place a new object in scene). There can be evaluated the use of memory and times of some specific nodes or the entire scene. On the project is used to take the information of the digital assets performances.

Useful Nodes and Contexts

During the development of the digital assets are required the knowledge of some specific nodes and contexts in Houdini.

Contexts

Object: This is the context that will be rendered and it includes lights, cameras and geometries.
SOP: (Surface Operators) in this context to create, deform and manipulate 3D objects is possible.
SHOP: (Shader Operators) this context creates the materials that will be applied to the 3D objects. Here the access to the Houdini’s data base is allowed with the material palette pane. Also to create new materials is allowed by creating a material node and dive into it to create the entire material.
CHOP: (Channel Operators) this context serves for manipulation of raw data such as motion and audio. Also animations’ modifiers can be implemented on this context.
Out: Here is where the user configures the render settings for the scene. Multiple renders are supported and the user can set up multiple profiles for these supported renderers.

Nodes

A sample of some of the nodes used during the digital asset design is described below, attaching a little description of the node’s function (for further information see the Houdini’s online help):

File: Allows reading or writing files.
Merge: Allows joining several geometries into just one. Used to display various objects at the same time or to paste some pieces of the same geometry
Copy: It can be used in two different ways. With just one input makes a specified number of copies of the input geometry. With two inputs, copy the first geometry into the points of the second geometry.
Transform: it is used to manipulate the input geometry. Permits to scale, rotate, translate and modify the input geometry or a group of the input geometry.
Captureproximity: It calculates the proximity between bones and points of the input geometry. The skeleton must be indicated on the parameter interface.
Polyreduce: Serves to reduce the quantity of polygons of the input geometry.
Deform: For a character context, uses an animated skeleton to deform the input geometry that was captured before by this type of skeleton.
Fuse: Allows reducing the number of points of a geometry by merging them using a distance threshold.
**Point:** In order to set some attributes to the points, this node allows changing or setting the value of the position, color, normals, etc.

**Object_merge:** allows importing on geometry level, geometries from other objects.

**Switch:** Allows varying the input of the following node by sorting its inputs and manipulating them with an input parameter.

**Mountain:** Generates a random noise deformation for the input geometry.

**Subdivide:** Divides the input geometries in order to have a smoother surface. It increases the number of primitives, points and polygons of the object.

**Ray:** Projects the left input into the right input. It is used to project the points into a plane.

**Material:** In the object context allows linking to the input geometry a material on the SHOP context. In the SHOP context is the container of the material attributes.

**Timeshift:** Serves to cook its input at a different time than it is cooking on. That permits access to earlier or later frames.

**Delete:** Is used to delete some points/primitives/details of the input geometry. It can be deleted by pattern, expression, bounding box or specific groups.

**Sort:** It serves to re-ordinate the point numbers of the input geometry. It can be sorted by point number or by an attribute.

**Add:** Allows creating points into the context, assigning to them a position and a weight.

**Scatter:** It creates an indicated number of points from the surface of the input geometry. Its output is the points.

**Edit:** Allows editing of the geometry by deforming single points/edges/primitives. It is controlled by handlers.

**Rop_geometry:** Serves to export a geometry or an animation to pre-calculate them and reduce the complexity of the scene.

**Trail:** Create trails behind the points.

**Attribcreate:** Allows creating a new parameter for the input geometry. Can be created at point, vertex, primitive or detail level.

**Attribtransfer:** Allows sharing the values of the attributes from 2 different geometries based on the distance between them.

**Attribcopy:** Allows copying the attributes from a geometry to another.

**Attribute:** Allows changing names or deleting attributes created before.

### Copy or Instancing

In order to make a great amount of copies of a single geometry Houdini provides two ways to do this. The first one is done at geometry level and it’s done with a copy node. The second one uses instances. For the first case a real geometry is copied and it will be always visible no matter the context where it is the actual work. For the second case moreover the geometry is only loaded or created at render time, so it won’t be visible in any moment.

Both of them have some advantages and disadvantages. The copy case introduces more complexity to the system but the copies are manipulable with the stamp variables which can be created as needed. Also the copy node assigns a univocal id to the copies, which is the copy number for the case of one input, or the point number for the case of two inputs. While instancing only brings the possibility of make changes at object level, it means that all the copies will be equals, just can be changed the context material, nevertheless it gives a lower complexity for the computer calculation.

### Attributes, Variables and Expressions

One of the most powerful tools that bring Houdini is the use of attributes and Python and Hscript expressions to control the animation development. The right uses of attributes permits to manage the creation and communication of its values to others geometries. The expressions permit to create behaviors, animations, movements, etc without the use of key framing.

A complete package of Hscript variables of different types can be found on Houdini. The Global variables are accessible from every part of the work; those variables are the Playbar Variables, the Global Variables, the Channel Variavbles, the COP-specific variables and the output driver specific
variables. On the other hand there are some Standard Variables that can be accessible or not depending on the context in which is working. These standard variables are used to describe objects (from vertex to details) by setting values for position, weight, bounding box and many others values that can be representative for an object.

On the SOP context a variable mapping can be done to move values from an attribute to a variable. During the design of digital assets this specific thing represents the soul of the animation, because it provides a programming environment over the node work flow. With the use of the expressions (that can be in one or both languages) those variables are read, calculated, manipulated and mixed in order to achieve the desire animation. Beyond the lecture of these variables, with the expressions the user is allowed to read information from specific channels of specific nodes, making a complex communication system within the animation. On the part 2 of this document will be explained one by one the expressions and the attributes used.

**Digital Asset Design**

Houdini Digital Assets are an extremely powerful way of extending Houdini. With these, you can package up all kinds of custom tools and hand them off to other users, or to yourself, for later use. You can hide the grimy details, so the user just deals with the problem as he would like to think of it. You can embed any arbitrary data, including scripts, images, etc.

Digital assets are custom operator types built from node networks. You can encapsulate the network in a digital asset, and then “promote” parameters from nodes inside the asset up as parameters or handles on the asset itself.

Digital assets let you capture the functionality of any network, as a reusable tool with a customized interface you can put in the hands of artists.

To use a digital asset is necessary to install operator type libraries (OTLs) into the context.

**Components**

A digital asset has several components that help you create digital assets while providing a high level interface for animators. These include:

<table>
<thead>
<tr>
<th>Subnet</th>
<th>A subnetwork operator is used to gather together all the parts of your digital asset and acts as the asset’s outer container. While working on an asset you will be able to dive into the subnet to set up and edit the asset. Later when an animator works with an asset, the operators inside the subnetwork are locked and cannot be edited directly.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nested Digital Asset</td>
<td>Digital assets can be nested within other digital assets to help break down a larger asset such as a character into more manageable parts. Each of these nested assets can have handles and controls that make it easier for animators to work with the parent asset.</td>
</tr>
<tr>
<td>Controls tab</td>
<td>When a subnet is turned into a digital asset operator, a new panel is added in the Parameter pane that contains all of those parameters promoted from the underlying network to the top level. These are the parameters that will be worked with by animators who are using the digital asset.</td>
</tr>
<tr>
<td>Handles</td>
<td>Interactive handles can be set up for a digital asset to provide direct manipulation within the Viewer pane for animators.</td>
</tr>
<tr>
<td>Type Properties</td>
<td>This panel is used by the technical director to create the digital asset controls and add help to the asset. This panel is locked down when the asset is put into the pipeline because the animator will focus on the Controls tab and any available handles.</td>
</tr>
<tr>
<td>Digital Asset Name</td>
<td>The name that is set in the Type properties plays an important role. It is this name that is used to ensure proper updating when newer versions of an asset are put into the pipeline. Different versions of an asset are usually given the same name but are managed through different Operator type libraries.</td>
</tr>
</tbody>
</table>

**Create a digital asset**

The Houdini’s online help brings a very good description step by step to create a digital asset:

1. Encapsulate the network of nodes that provide the asset’s functionality inside a subnet.
Anything being referenced by the operators inside your subnet must either be moved into the subnet or re-referenced through the asset’s interface.

If you decide to put the linked nodes inside the subnet you will need to use network managers to mix network types. For example, if you put an object into a subnet that links to shaders under /shop, you must create a SHOP network manager inside your subnet and move your shader into the manager.

1. Press RMB on the subnet node and choose **Create Digital Asset**.
2. In the window that appears, give your new asset a unique internal name and a human-readable label. You can change the label later but the name cannot be changed after the asset is created.
3. Use the sliders to set the minimum and maximum number of inputs to your new asset. In some cases, inputs may have been created when you collapsed your operators into a subnet.

An asset with minimum inputs of 0 means the node doesn’t need to have an input connected to be valid. For object-level assets, this means the node doesn’t have to be parented.

4. Use the **Save to library** field to choose an OTL file to save the new asset into.

While it’s possible to save multiple assets into a single library file, Side effects recommend you keep each asset in its own .otl file, and name it the same as the library. This makes managing assets in the pipeline much easier.

By default $HOME/opCustom.otl is filled in. This is a good place to put assets when you are first learning.

If you enter the special string Embedded instead of a filename, the asset will be saved in the .hip file instead of in a library.

5. Use the **Install Library To** option to choose where you want the type library to be available.
6. Click **Accept**. Houdini saves the new asset into the OTL file you chose.
7. Houdini opens the type properties window to allow you to edit more properties of the new asset, including the creation of the parameter interface.

**Parameter Interface**

The parameter interface will be the only place where a digital asset could be modified in the future. Basically this parameter interface will content promoted parameters from the inner nodes, namely the parameters that are externally visible will affect directly parameters from the network inside the package. On the type properties window select the Parameters tab. Then on the first frame that is called create parameters select the “From nodes” tab. Then simply search the interested parameter and drag it to the second frame. The existing parameters tab allows organizing the different types of parameters. Using the parameters created by type, an organized on tabs and very good looking parameter interface can be achieved. The third frame is called parameter description, on this section can be defined the type of data, the range of the input for the parameters and by scripting can be described more important features, like disable the parameter when other has a determined value.

Also new parameters can be created in order to modify some values at the same time inside. To achieve that it’s necessary on the parameter description indicates which values are referred to this parameter.
# Appendix B – Attributes and Parameter Reference

## ATTRIBUTES TABLE

<table>
<thead>
<tr>
<th>Node</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>1. Points_creation</strong></td>
</tr>
<tr>
<td>Attribcreate_speedz</td>
<td>speed1</td>
<td>Speed in z (from CHOP)</td>
</tr>
<tr>
<td>Attribcreate_speedx</td>
<td>speed2</td>
<td>Speed in x (from CHOP)</td>
</tr>
<tr>
<td>Attribcreate_rot</td>
<td>rot</td>
<td>Rotation of the animation (from CHOP)</td>
</tr>
<tr>
<td>Attribcreate_idx</td>
<td>idx</td>
<td>Id of the animation (From copy1/stamp)</td>
</tr>
<tr>
<td>Attribcreate_aaa</td>
<td>aaa</td>
<td>Id of the point (From copy1/stamp)</td>
</tr>
<tr>
<td>Attribcreate_col2</td>
<td>col</td>
<td>For collision. To use after.</td>
</tr>
<tr>
<td>Attribcreate_speed</td>
<td>speed</td>
<td>Max Value between speed1 and speed2.</td>
</tr>
<tr>
<td>Attribcreate_oil</td>
<td>oi</td>
<td>Initial offset of the point (from placement)</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>2. Delete_points and Delete_initial_Obstacles</strong></td>
</tr>
<tr>
<td>Attribcreate_col</td>
<td>col</td>
<td>Collision Flag</td>
</tr>
<tr>
<td>Attribcreate4</td>
<td>rmo</td>
<td>Obstacle Flag</td>
</tr>
<tr>
<td>Attribcreate_rmoo</td>
<td>rmoo</td>
<td>Mutual Obstacle Flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>3. Obstacles_Avoidance</strong></td>
</tr>
<tr>
<td>Attribcreate_obs</td>
<td>Obs</td>
<td>Obstacle flag</td>
</tr>
<tr>
<td>Attribcreate_rad</td>
<td>Rad</td>
<td>Avoidance Ellipse X Radio</td>
</tr>
<tr>
<td>Attribcreate_radz</td>
<td>Radz</td>
<td>Avoidance Ellipse Z Radio</td>
</tr>
<tr>
<td>Attribcreate_repos</td>
<td>Relpos[3]</td>
<td>[Left/Right, Back/In Front, Distance]</td>
</tr>
<tr>
<td>Attribcreate_refe</td>
<td>refe</td>
<td>Reference position</td>
</tr>
<tr>
<td>Attribcreate_alfa</td>
<td>alfa</td>
<td>Offset to increase the X axis position</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>4. Attributes_creation_for_Avoidance and distance</strong></td>
</tr>
<tr>
<td>Attribcreate_fc</td>
<td>fc</td>
<td>Collision flag $F+1</td>
</tr>
<tr>
<td>Attribcreate_fca</td>
<td>fca</td>
<td>Collision flag $F</td>
</tr>
<tr>
<td>Attribcreate_aux</td>
<td>aux[4]</td>
<td>[id of the point, X, Z, Speed]</td>
</tr>
<tr>
<td>Attribcreate3</td>
<td>dist</td>
<td>Distance between points</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>5. Avoidance</strong></td>
</tr>
<tr>
<td>Attribcreate_radio</td>
<td>rd</td>
<td>Avoidance radio</td>
</tr>
<tr>
<td>Attribcreate_flag</td>
<td>flag</td>
<td>Speed comparison</td>
</tr>
<tr>
<td>Attribcreate_flag2</td>
<td>flagc</td>
<td>X position comparison</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>Attribcreate_flag1</td>
<td>flagb</td>
<td>Z position comparison</td>
</tr>
<tr>
<td>Attribcreate_beta</td>
<td>beta</td>
<td>Offset to increase the X axis position</td>
</tr>
</tbody>
</table>

**PARAMETER REFERENCE**

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Referenced Node</th>
<th>Parameter referred</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Help</td>
<td>Help/switch1</td>
<td>Select Input</td>
</tr>
<tr>
<td>View Ground</td>
<td>View_World/Switchground</td>
<td>Select Input</td>
</tr>
<tr>
<td>Ground</td>
<td>Ground/switch1</td>
<td>Select Input</td>
</tr>
<tr>
<td>Ground Path</td>
<td>Ground/YOUR_TERRAIN</td>
<td>Object path</td>
</tr>
<tr>
<td>Ground Scale</td>
<td>Ground/xform1</td>
<td>Uniform scale</td>
</tr>
<tr>
<td>Ground Translate</td>
<td>Ground/xform1</td>
<td>Rotate vector</td>
</tr>
<tr>
<td>Ground Rotate</td>
<td>Ground/xform1</td>
<td></td>
</tr>
<tr>
<td>World Scale</td>
<td>Move_v2</td>
<td>Scale</td>
</tr>
<tr>
<td></td>
<td>Obstacles</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Obstacles1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ground</td>
<td></td>
</tr>
<tr>
<td>View Obstacles</td>
<td>View_World/Switchobstacles</td>
<td>Select input</td>
</tr>
<tr>
<td>Number of obstacles</td>
<td>Obstacles(1)/scatter1</td>
<td>Number of points</td>
</tr>
<tr>
<td>Obstacles</td>
<td>Obstacles(1)/switch1</td>
<td>Select Input</td>
</tr>
<tr>
<td>Obstacles path</td>
<td>Obstacles(1)/LOAD_OBSTACLES</td>
<td>Object path</td>
</tr>
<tr>
<td>Default</td>
<td>Obstacles(1)/switch2</td>
<td>Select input</td>
</tr>
<tr>
<td>Placement Path</td>
<td>Obstacles(1)/object_merge1</td>
<td>Object path</td>
</tr>
<tr>
<td>Placement</td>
<td>Move_v2/switch2</td>
<td>Select input</td>
</tr>
<tr>
<td>Placement scale x</td>
<td>Move_v2/xform5</td>
<td>Scale x</td>
</tr>
<tr>
<td>PLACEMENT</td>
<td>Move_v2/LOAD_PLACEMENT</td>
<td>Object path</td>
</tr>
<tr>
<td>Number of agents</td>
<td>Move_v2/copy1</td>
<td>Number of copies</td>
</tr>
<tr>
<td>Human Scale</td>
<td>Move_v2/xform1</td>
<td>Scale</td>
</tr>
<tr>
<td>Human Geo</td>
<td>Human_geometry/switch1</td>
<td>Select input</td>
</tr>
<tr>
<td>Load Geometry</td>
<td>Human_geometry/file2</td>
<td>file</td>
</tr>
<tr>
<td>Load Material</td>
<td>Human_geometry/material2</td>
<td>SHOP</td>
</tr>
<tr>
<td>Material</td>
<td>Human_geometry/switch2</td>
<td>Select input</td>
</tr>
<tr>
<td>----------</td>
<td>------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Polygons</td>
<td>Human_geometry/polyreduce1</td>
<td>Percentage</td>
</tr>
<tr>
<td>ANIMATIONS DIR.</td>
<td>CHOP_FILE_MASTER</td>
<td>File</td>
</tr>
<tr>
<td>Min Human Distance</td>
<td>Move_v2/attribtransfer1</td>
<td>Distance threshold</td>
</tr>
<tr>
<td>Human Object Initial</td>
<td>Move_v2/attribtransfer4</td>
<td>Distance threshold</td>
</tr>
<tr>
<td>Human Object Avoidance</td>
<td>Move_v2/attribtransfer6</td>
<td>Distance threshold</td>
</tr>
<tr>
<td>Avoidance x Radio</td>
<td>Move_v2/attribcreate_rad</td>
<td>Value 1</td>
</tr>
<tr>
<td>Avoidance z Radio</td>
<td>Move_v2/attribcreate_radz</td>
<td>Value 1</td>
</tr>
<tr>
<td>2 Avoidance x Radio</td>
<td>Move_v2/attribcreate_rad</td>
<td>Value 2</td>
</tr>
<tr>
<td>2 Avoidance z Radio</td>
<td>Move_v2/attribcreate_radz</td>
<td>Value 2</td>
</tr>
<tr>
<td>Human Avoidance Dist.</td>
<td>Move_v2/attribtransfer2</td>
<td>Distance threshold</td>
</tr>
<tr>
<td>Human Avoidance Radio</td>
<td>Move_v2/attribtransfer3</td>
<td></td>
</tr>
<tr>
<td>Render Button Human</td>
<td>Move_v2/rop_geometry1</td>
<td>Start Render</td>
</tr>
<tr>
<td>Valid Frame Range (h)</td>
<td>Move_v2/rop_geometry1</td>
<td>List</td>
</tr>
<tr>
<td>Start/End/Inc</td>
<td>Move_v2/rop_geometry1</td>
<td>Vector</td>
</tr>
<tr>
<td>Output File</td>
<td>Move_v2/rop_geometry1</td>
<td>File Path</td>
</tr>
<tr>
<td>Render Button Human</td>
<td>View_world/rop_geometry1</td>
<td>Start Render</td>
</tr>
<tr>
<td>Valid Frame Range (h)</td>
<td>View_world/rop_geometry1</td>
<td>List</td>
</tr>
<tr>
<td>Start/End/Inc</td>
<td>View_world/rop_geometry1</td>
<td>Vector</td>
</tr>
<tr>
<td>Output File</td>
<td>View_world/rop_geometry1</td>
<td>File Path</td>
</tr>
</tbody>
</table>