Chapter 2

Character Animation Framework

2.1 Introduction

One of the fundamental problems concerning the creation of fully-autonomous animated characters is the design of an overall motion-control software framework. Our approach is to view the problem from a robotics perspective, drawing upon the tools and techniques used in the design and control of actual robots. Such tools include algorithms from computational geometry, artificial intelligence, computer vision, and dynamic control systems. The animated character is viewed as an *autonomous agent* in a virtual environment (i.e. a "virtual robot")

2.2 Autonomous Agents

Background

The concept of an *autonomous agent* has received a fair amount of attention in the recent artificial intelligence literature. Indeed, autonomous agents have been touted as representing "a new way of analysing, designing, and implementing complex software systems"[JSW98]. Ironically, even though several international conferences are held annually on the topic, the term *agent* lacks a universally accepted definition (for a summary discussion, see [FG97]). The word *autonomy* is similarly difficult to define

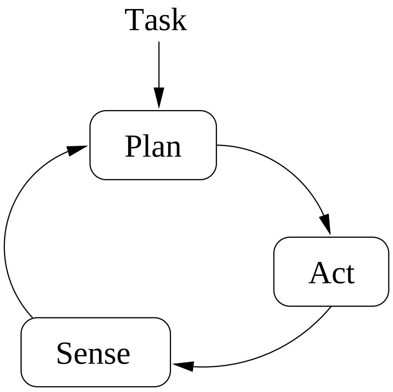


Figure 2.1: The control loop of a robot (physical agent).

precisely. In the context of animated characters, we will consider *autonomous* to mean not requiring the continuous intervention of a user, and adopt the following general definition as a starting point:

> "An *autonomous agent* is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future."
>
> Franklin & Graesser, 1996
> *"Is it an Agent, or just a program?"*
> Proc. of 3rd. Int. Workshop on Agent Theories

Physical Agents

A physical agent (what we commonly conceive of as a "robot") is essentially a collection of hardware and software that operates in the physical world. It has control inputs (motors) that apply forces and torques on its joints, which in turn cause the robot to move itself or other objects in it environment. The robot also comes equipped with various kinds of sensors that provide feedback on its motion, along with information about its surroundings. Periodically, the robot gathers input from its sensors and formulates a plan of action based upon its current goals and internal state. The plan of action is converted into control inputs, and the cycle repeats.

Figure 2.1 shows the control loop of a generalized physical agent. In reality, the underlying architecture may be much more complex, with potentially several servo or high-level planning control loop modules executing in parallel. However, the overall operation is essentially the same. Given a program of tasks, the robot utilizes the cycle of sensing, planning, and acting, in an effort to accomplish its goals. If the robot has been well-designed and programmed, it will correctly perform its given tasks.

Animated Agents

An *animated agent* is a software entity that exists and moves within a graphical virtual world. We consider the problem of motion generation for an animated agent as essentially equivalent to that of *designing and controlling a virtual robot*. Instead of operating in the physical world, an animated agent operates in a simulated virtual world, and employs a "virtual control loop". The virtual control loop operates as follows:

Control Inputs: The animated agent has a general set of *control inputs* which define its motion. These may be values for joint variables that are specified explicitly, or a set of forces and torques that are given as input to a physically-based simulation of the character's body.

Virtual Sensors: The animated agent also has a set of "virtual sensors" from which it obtains information about the virtual environment.

Planning: Based on information provided by the virtual sensors and the agent's current tasks and internal state, appropriate values for its control inputs are computed.

The animated agent exists entirely as a software program that runs in a simulated virtual environment. This environment is periodically rendered on a graphic display to produce a series of images. The viewer of the display (the user) interprets the sequence of images as apparent motion of the objects and characters within the virtual environment.

In some cases, the user is capable of directly interacting with the simulation through input devices. This distinguishes the situation from the passive act of simply viewing a television or movie screen, where the information flow is unidirectional (from the screen to the user). The animated agent can respond to the interactions of the user, as it goes about attempting to fulfill its agenda of tasks.

The advantages that a virtual robot has over a physical robot are numerous. While a physical robot must contend with the problems of uncertainty and errors in sensing and control, a virtual robot enjoys "perfect" control and sensing. As an aside, this means that it should be *easier* to design an animated agent that behaves intelligently, than a physical agent that does so. In fact, creating an intelligent autonomous animated agent can be viewed as a necessary step along the way towards creating an intelligent autonomous physical robot. If we cannot create a robot that behaves intelligently in simulation, how can we expect to create one in the real world? Thus, in some sense, this research goes beyond just computer animation, but also extends into the realm of artificial intelligence.

2.3 Motion Synthesis

Animation Variables

Let us now assume that we are faced with the problem of generating motion for an animated character given a task command. With so many possible ways for a character to move, how can a software program select from among the near infinite choices available to complete a given task? The difficulty of this problem generally increases with the complexity of the character. Human-like characters have a large number of animation variables (*avars*), or degrees of freedom (DOF) that must be specified. The variables that control the positions of the parts of the human body model are sometimes referred to as the *joint variables*. These variables typically control approximately 15 to 60 movable parts arranged hierarchically, with anywhere between 40 to 110 degrees of freedom (e.g. the model shown in Figure 2.2). This total does not include the detailed degrees of freedom in the face, which can number in the

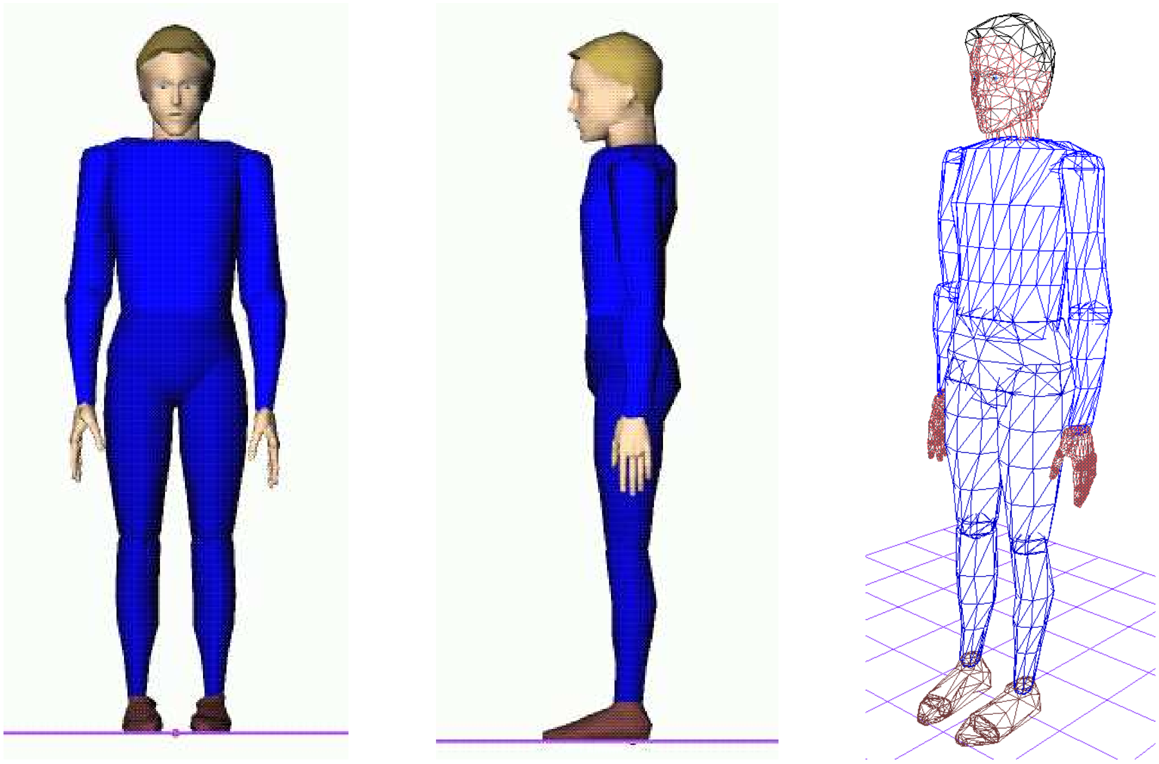
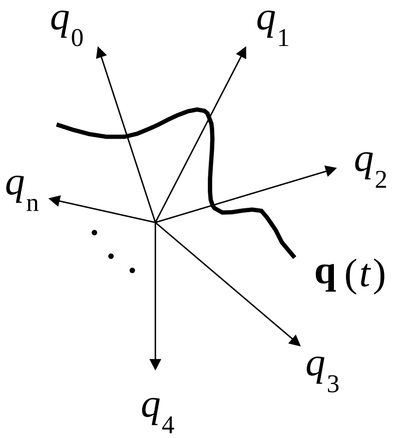


Figure 2.2: Multiple views of a human character model composed of 52 moving parts with a total of 104 degrees of freedom

hundreds depending upon the complexity of the face model.

Mathematically, the animation variables for a character together form a parameter space of possible *configurations* or *poses*. From a robotics standpoint, this space can be considered the *joint space*, or the *configuration space* ($\mathcal{C}$-space) of the character[LP83]. Any motion for the character will trace out a curve (i.e. a *path*) in this multi-dimensional space as illustrated in Figure 2.3. The curve is normally a continuous function of time (i.e. a *trajectory*). In addition, the topology of the space may be multiply connected so as to reflect the fact that certain variables "wrap around" in value (e.g. global orientation). Conceptually, the fundamental goal of any motion synthesis strategy is *to generate a trajectory in the configuration space that accomplishes the desired task*. In addition to this basic requirement of accomplishing the task, other important criteria may include generating smooth motion, physically-valid motion, or simply motion that is perceived by a user as generally aesthetically-pleasing. This



$$\mathbf{q}(t) = \begin{pmatrix} q_0(t) \\ q_1(t) \\ \vdots \\ q_n(t) \end{pmatrix}$$

$$t \in [0, t_{max}]$$

Figure 2.3: Motions for a character trace out a time-parameterized curve in the multi-dimensional joint space.

last criterion is perhaps the most important for computer animation, yet also the most difficult to satisfy, since "aesthetics" cannot be readily defined mathematically or objectively.

Classes of Motion

The motions that we may wish to generate for animated characters tend to fall into two categories:

Primary Motions: These include gross body movements (e.g. walking, jumping, reaching), as well as facial animation (e.g. speech, expressions). Primary motions are sometimes referred to as *active* or *intentional* motions, since they often involve joints that are under a character's direct control. Thus, they are driven by a character's willful (intentional) acts.

Secondary Motions: These include the animation of hair, skin, clothing, or motion related to environmental effects (e.g. smoke from a character's cigarette). Secondary motions are sometimes referred to as *passive* or *non-intentional* motions, since they involve animation variables that are typically not under a character's direct control. Instead, the motions are an indirect result of the character's actions (e.g. the wrinkling of a character's shirt in response to a reaching motion).

The approaches taken to solving motion generation problems among these two classes of motion will often differ. Some motion synthesis strategies are more suited to primary motions (e.g. motion planning), while other are more readily useful for creating secondary motions (e.g. physically-based modeling).

Facial animation is a highly-specialized topic with its own unique challenges, and is beyond the scope of this thesis. However, as any skilled animator will testify, facial animation is a vitally important aspect to creating believable action and drama. Thus, any serious automatic motion generation software should support the animation of the face.

Within the remaining chapters of this thesis, we will focus our attention on the automatic generation of gross body motions, specifically that of goal-directed navigation (walking and running), and object grasping and manipulation.

2.4 High-Level Tasks

Specification of Task Commands

We will consider the generation of motion for human-like animated characters that can respond to task-level commands. such as "walk to the kitchen", "open the refrigerator", "take out the milk", "pour a glass", "sit down at the table", "take a drink", "wave hello to Mr. Smith", or "move the teapot next to the Luxo lamp in front of the ray-traced sphere". Task commands may be given to a character in a number of ways:

User-specified: Task commands generated explicitly by a user through keystrokes, mouse clicks, voice commands, or other direct means.

Software scripts: High-level behavior programs specified via combinations of typical programming language constructs, such as IF-THEN-ELSE, WHILE-DO, REPEAT, etc.

Behavior simulations: Software that attempts to model the physical, mental, or emotional state of the character, and generates task-level commands based on its internal goals, needs, desires, or preferences. This approach may be considered a special case of a software script approach that employs a detailed model of the character's internal functions or thought processes.

Given a task command (or a set of high-level goals) provided by one of the above sources, the software must automatically generate the motion for the character to complete the task. From here on, we shall refer to this problem as the *motion synthesis problem*, and the corresponding software as the *motion synthesis software*.

Desired Motion

From any given task command, the input and the output are essentially of the same form. The input given is:

1. A geometric description of the environment.

2. A kinematic and geometric description of the character (possibly including joint limits, or physical attributes such as mass, moments of inertia, etc., for each link).

3. The *initial state* of the character (the initial values for each joint parameter of the character, and any auxiliary animation variables).

4. The *goal state* of the character (either directly specified or computed from the task command).

The desired output will be a specification for each of the character's joint variables expressed as a continuous function of time. The simultaneous playback of all of the joint trajectories will cause the character to move so as to accomplish the given task.

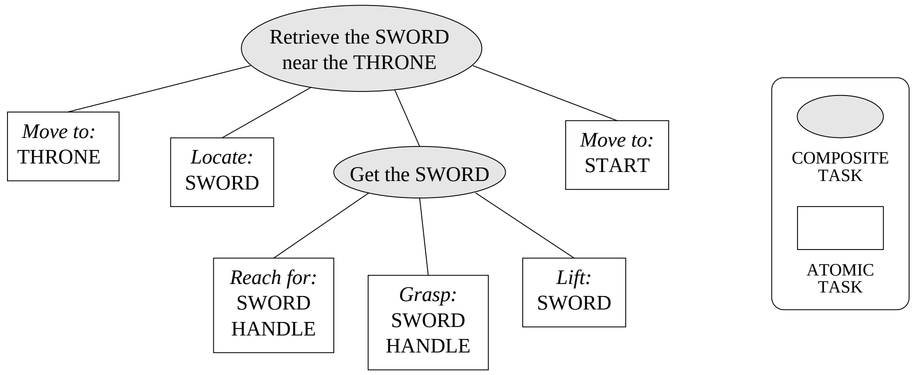


Figure 2.4: A composite task broken down into a linear sequence of atomic tasks

Composite Tasks and Atomic Tasks

The ability to synthesize motion for a character depends upon restricting the domain of task commands allowable. Therefore, some task commands can be considered as basic *atomic tasks* such as “move to a location” or “grasp an object”. Atomic tasks such as these have final goal states which are generally less ambiguous, or may have default parameters specified to allow a fast and fairly straightforward computation of the goal state.

From these atomic task primitives, one can build up more complicated *composite* task commands. For example “retrieve the sword near the throne” may be decomposed into the following sequence of sub-tasks: “move to the throne”, “locate the sword”, “get the sword”, and “move to the starting location” (see Figure 2.4). The sub-task “get the sword” may be further subdivided into the tasks “reach towards the sword handle”, “grasp the sword handle”, and “lift the sword”. Each sub-task will generally involve an initial and a goal state which is less ambiguous, and therefore simpler to compute. The goal state of one sub-task will become the initial state of the subsequent sub-task. Structuring task commands based on a hierarchy of sub-tasks allows one to perform a recursive decomposition until a linear chain of atomic task commands is obtained. To complete the composite task, the character needs only to complete all atomic sub-tasks in the chain.

2.5 A Virtual Robot Architecture

Overview

If every atomic task was very narrowly and explicitly defined, one could imagine simply maintaining a vast library of pre-computed, captured, or hand-animated motions to accomplish every possible task. The reality is that, in general, tasks cannot be so narrowly defined, lest the set of possible tasks become infinite. Instead, atomic tasks are typically specified at a high-level, and apply to a general class of situations, such as "move to a location" or "grasp an object". Thus, they are specific in one sense (the required end result), but also very vague (how the result is achieved).

At present, no single motion synthesis strategy can adequately handle the variety of tasks that can arise. Instead, a flexible underlying software framework is needed to bring together different tools and techniques. As a foundation, we propose an animated agent framework based on a virtual control loop as outlined in Section 2.2. At regular intervals, the agent computes a motion strategy based on the current task, its internal state, and its perceived model of the environment. The motion is generated using a collection of fundamental software modules and data libraries depending upon the task.

Some possible software components include a large library of "canned motions" (clip motions), a simulated sensing module, a physically-based simulation module, and a library of motion planning and geometric algorithms. Other potential component modules include numerical optimization libraries, inverse kinematics routines, and collections of biomechanical or neurophysiological data appropriate to specific classes of character morphology (e.g. humans, dogs, birds). As an example, Chapter 3 describes a goal-directed navigation strategy that combines cyclic motion capture data, a path planning module, and a path following controller. Chapter 4 extends this technique to incorporate a simulated sensing module. Chapter 5 describes a motion strategy for object grasping and manipulation that utilizes an inverse kinematics module and a randomized path planner and optimizer. What is important is that different software modules are accessible for use by different motion synthesis strategies depending upon the task.

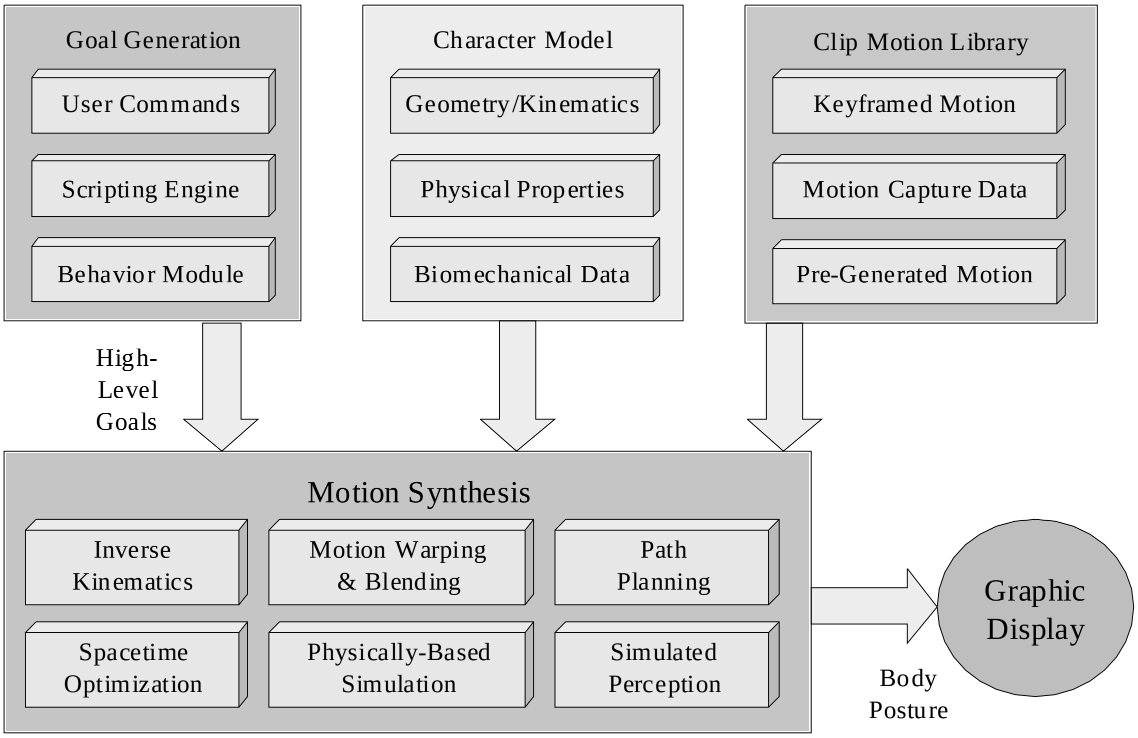


Figure 2.5: Resources available for motion synthesis.

A block diagram of the agent's available software resources is depicted in Figure 2.5. A set of high-level goals are generated and passed to the motion synthesis software. Utilizing some or all of the fundamental software components and data libraries, the resulting motion for the character is computed and passed to the graphic display device for rendering.

Mathematical Formulation

To make things more precise, we now give a more formal formulation of the motion synthesis problem for animated characters. The notation adopted here is loosely based on the conventions used in [Lat91], which are often used in the robotics and motion planning literature.

1. A character or *agent* is called $\mathcal{A}$. If there are several characters, they are called $\mathcal{A}_i$ $(i = 1, 2, \ldots)$.

2. The 3D environment in which the characters move is denoted by $\mathcal{W}$ (commonly called the *workspace* in robotics), and is modeled as the Euclidean space $\Re^3$ ($\Re$ is the set of real numbers).

3. Each character $\mathcal{A}$ is a collection of p links $\mathcal{L}_j$ ($j = 1, \ldots, p$) organized in a kinematic hierarchy with cartesian frames $\mathcal{F}_j$ attached to each link.

4. A *configuration* or *pose* of a character is denoted by the set $\mathcal{P} = \{T_1, T_2, \ldots, T_p\}$, of p relative transformations for each of the links $\mathcal{L}_j$ as defined by the frame $\mathcal{F}_j$ relative to its parent link's frame. The *base* or *root* link transformation T_1 is defined relative to some world cartesian frame $\mathcal{F}_{world}$.

5. Let n denote the number of generalized coordinates or *degrees of freedom* (DOFs) of $\mathcal{A}$. Note that n is in general *not equal* to p. For example, a simplified human arm may consist of three links (upper arm, forearm, hand) and three joints (shoulder, elbow, wrist) but have seven DOFs ($p = 3$, $n = 7$). Here, the shoulder and the wrist are typically modeled as having three rotational DOFs each, and the elbow as having one rotational DOF, yielding a total of seven DOFs.

6. A *configuration* of a character is denoted by $\mathbf{q} \in \mathcal{C}$, a vector of n real numbers specifying values for each of the generalized coordinates.

7. Let $\mathcal{C}$ be the *configuration space* or $\mathcal{C}$-space of the character $\mathcal{A}$. $\mathcal{C}$ is a space of dimension n.

8. Let $\textsc{Forward}(\mathbf{q})$ be a *forward kinematics* function mapping values of $\mathbf{q}$ to a particular pose $\mathcal{P}$. $\textsc{Forward}(\mathbf{q})$ can be used to compute the global transformation G_j of a given link frame $\mathcal{F}_j$ relative to the world frame $\mathcal{F}_{world}$.

9. Let $\textsc{Inverse}(\mathcal{P})$ be a set of *inverse kinematics* (IK) algorithms which maps a given global transformation G_j for a link frame $\mathcal{F}_j$ to a set $\mathcal{Q}$ of values for $\mathbf{q}$. Each configuration $\mathbf{q} \in \mathcal{Q}$, represents a valid inverse kinematic solution ($\textsc{Forward}(\mathbf{q})$ positions the link $\mathcal{L}_j$, such that the frame $\mathcal{F}_j$ has a global transformation of G_j relative to $\mathcal{F}_{world}$). Note that the set $\mathcal{Q}$ may possibly be infinite, or the empty set (no valid solutions exist).

10. The obstacles in the environment $\mathcal{W}$ are denoted by $\mathcal{B}_k$ $(k = 1, 2, \ldots)$.

11. We define the *$\mathcal{C}$-obstacle region* $\mathcal{C}B \subset \mathcal{C}$ as the set of all configurations $\mathbf{q} \in \mathcal{C}$ where one or more of the links of $\mathcal{A}$ intersect (are in collision) with another link of $\mathcal{A}$, any of the obstacles $\mathcal{B}_k$, or any of the links of another character $\mathcal{A}_i$ with $\mathcal{A}_i \neq \mathcal{A}$. We also regard configurations $\mathbf{q} \in \mathcal{C}$ where one or more *joint limits* are violated as part of the $\mathcal{C}$-obstacle region $\mathcal{C}B$.

12. The open subset $\mathcal{C} \setminus \mathcal{C}B$ is denoted by $\mathcal{C}_{free}$ and its closure by $cl(\mathcal{C}_{free})$, and it represents the *space of collision-free configurations* in $\mathcal{C}$ of the character $\mathcal{A}$.

13. Let $\tau : \mathcal{I} \mapsto \mathcal{C}$ where $\mathcal{I}$ is an interval $[t_0, t_1]$, denote a motion trajectory or *path* for a character $\mathcal{A}$ expressed as a function of time. $\tau(t) = \mathbf{q}_t$ represents the configuration $\mathbf{q}$ of $\mathcal{A}$ at time t, where $t \in \mathcal{I}$.

14. A trajectory τ is said to be *collision-free* if $\tau(t) \in \mathcal{C}_{free}$ for all $t \in \mathcal{I}$.

In response to an atomic task command for a character, we wish to compute a motion trajectory τ that satisfies certain criteria. At a minimum, the resulting motion of $\mathcal{A}$ should connect the initial configuration $\mathbf{q}_{init}$ and the goal configuration $\mathbf{q}_{goal}$. Though not strictly required, it is also helpful if the motion trajectory is *collision-free* (the character's geometry does not pass through objects in the environment). In other words, τ should be computed such that:

1. $\tau(t_0) = \mathbf{q}_{init}$ for the start time t_0.

2. $\tau(t_1) = \mathbf{q}_{goal}$ for some final time t_1.

3. $\forall t \in [t_0, t_1]$, $\tau(t) \in \mathcal{C}_{free}$.

The forward kinematics function $\textsc{Forward}(\mathbf{q})$ can be used to map intermediate configurations defined by τ to the corresponding character poses used during graphic rendering and display. A software module that computes τ for specific task commands is called a *motion synthesis strategy*.

There are several criteria which are important for the purpose of evaluating any proposed motion synthesis strategy.

1. *Computational efficiency*: Given the available computing resources, how much time is required to compute a motion? Is this strategy practical for the desired application?

2. *Reliability and consistency*: Do the algorithms always terminate? Do they find valid solution trajectories whenever they exist? Do similar task queries yield similar results?

3. *Quality of the motion*: Is the generated motion physically valid? Does it appear natural? Does it reflect the personality or unique aspects of the character? Overall, is it aesthetically pleasing?

These criteria (especially naturalness and aesthetics) are more difficult to define mathematically, and typically must be evaluated by viewing the output motions resulting from repeated trials.

In designing a motion synthesis strategy, a variety of general-purpose software tools (*software modules*) are available for use. Different modules will be more effective and useful for certain classes of task commands. In addition, typically one or more modules can be used in combination in order to solve specific motion synthesis problems.

The Roles of Specific Software Modules

This section describes a few of the fundamental software components identified in Figure 2.5, and briefly indicates how they might be utilized for synthesizing motion. This is by no means an exhaustive list, but adequate for creating interesting animations (see Chapter 6). Again, we believe that no single component can provide a general solution to the motion synthesis problem, but rather each technique in combination with one or more of the others can provide a viable approach to generating animation for a given set of tasks.

Clip Motion Libraries

Clip motions or "canned motions" are short animations that have been either keyframed, pre-generated, or obtained via motion capture systems. Such motions can be stored and played back very efficiently. Clip motions obtained via motion capture systems are recorded directly from a live subject, and then applied to a character for the purposes of animation. Captured motions are often preferred due to their high level of visual realism. This is especially true of motions which contain high-frequency components in the data (e.g. rapid gestures), or motions in which a character's particular body mannerisms (style of motion) must be reproduced.

Large libraries of clip motions can potentially become a powerful resource for animation. Eventually, animating a character in a given situation could ultimately involve selecting a pre-recorded motion from a vast *motion dictionary* indexed by task or motion characteristics. For example, there may be hundreds of walking motions stored, from among which a character might utilize a "medium-fast walk with a slight limp in the left leg".

As mentioned in Chapter 1, the primary drawback to clip motions is that any given data set can usually only be used in a very specific set of situations. For example, consider a captured motion of a human character opening a refrigerator and taking out a carton of milk. The motion will only appear perfect if the virtual model of the character, the refrigerator, the carton, and their relative positions match the actual objects used when the motion was captured. What happens if the carton is placed on the lower shelf instead of the upper shelf? What if the refrigerator model is made larger, or the location of the handle on the refrigerator door changes? What happens if a bottle of orange juice is placed in front of the milk carton? Motion warping, interpolation, or spacetime constraint techniques exist that adapt clip motions to a broader class of situations while enforcing kinematic and/or dynamic constraints[WP95, Gle98, BB98, PW99]. However, larger deviations can often cause such adapted motions to lose their visual realism, and hence their aesthetic quality. Finding new techniques to adapt individual motions and connect sequences of clip motions is an active area of research.

Captured motions are most useful for *efficiently animating activities that would* *otherwise be very difficult to model in software* (e.g. sitting in a chair, dancing, or animating human speech and facial expressions). The human eye is very keen at detecting subtle flaws in human motion animation. Thus, most software simulations have so far been unable to match the pure visual realism obtained by simply using captured motions. For this reason clip motions will continue to play an important role in real-time animation systems.

Motion Planning

Motion planning algorithms were initially developed in the context of robotic systems. Such algorithms generate motion given a high-level goal and a geometric description of the objects involved[Lat91]. In the context of computer animation, motion planning can be used to effectively compute primary (intentional) motions. Examples include computing collision-free motions to accomplish high-level navigation or object manipulation tasks[KKKL94a, Ban98, Kuf98], or connecting different body configurations[CB92, BMT97]. Motion planning is particularly suited to such tasks, since there is a near infinite number of possible goal locations and obstacle arrangements in the environment. This combinatorial explosion of possibilities currently prohibits the direct use of pre-recorded motion sequences. Instead, flexible and efficient planning algorithms can be designed to compute collision-free motions for specific categories of tasks commands that apply to a broader set of situations (see Chapter 3 and Chapter 5).

However, motion planning is less useful for tasks in which few obstacles to motion exist and/or aesthetics are of paramount importance (such as facial animation). As an example, consider the task of animating a human figure sitting down on a chair. This task results in a highly-specialized class of motions involving a direct interaction of the entire body with an environment object (in this case, a chair). Grasping the armrests during the sitting motion and adjusting the body posture afterwards are subtleties of motion that are important for naturalness, and are unlikely to be obtained by directly searching the body configuration space. In this case, using motions derived from captured data can potentially yield much better results, when compared with attempts to construct complicated kinematic or physically-based models, or apply motion planning techniques.

The primary challenge when using motion planning to generate animation is to design efficient algorithms that achieve visual realism. Aesthetics of motion are of little concern for robots, but are vitally important for animated characters. The computed motion must look natural and realistic. It may be possible to encode aesthetics as search criteria to use during planning, or to perform post-processing on the planned motion. For example, the naturalness and realism of a planned motion could arise from an underlying physically-based model that guides the search. Alternatively, search criteria might be ultimately derived from clip motion libraries that represent a particular "style" of motion.

Physically-Based Simulation

All motions in the physical world are driven by the laws of physics. Motions in virtual worlds typically aspire to give the appearance that they are also driven by the laws of physics. Graphical models simulate the visual appearance of objects, while physical models simulate their behavior in the physical world.

Animation generated using physically-based models (dynamic simulation) has the potential advantage of exhibiting a very high level of realism. Given a set of initial conditions and force/torque models, the motions of objects and their interactions can be calculated precisely. However, since the underlying motion is dictated by physics, it is difficult to control the simulation at a task-level. Spacetime constraint optimization techniques can alleviate some of these difficulties[IC88, WM88, LGC94], but at a computational cost that is largely prohibitive for interactive animation systems.

Physically-based techniques are very well-suited for generating secondary motions. Examples include the animation of rigid objects[AG85, Bar89, Mir96], hair, skin, and clothing[TF88, VCNT95, BW98], or environmental effects (wind, water, smoke, fire)[SOH99]. Animating special effects whose motion derives primarily from the force of gravity (e.g. a human character falling from a bicycle or slipping on a banana peel) are ideal situations where physically-based simulations can be applied.

However, it is more difficult to apply such techniques to the animation of primary (intentional) motions, using a physically-based model of a character. For example, consider tasks in which the dynamics of the underlying physical system (e.g. the human body) plays a significant role in the overall quality of the motion (e.g walking, dancing, running, jumping, kicking, etc). Fundamentally, the key difficulty lies in computing the required controls necessary to achieve a particular task. Some complex controllers have been carefully hand-designed for animating human walking, running, and athletic maneuvers[MZ90, RH91, Hod96, HW98]. Other researchers have proposed automatically synthesizing controllers via dynamic programming[dPFV90], genetic algorithms[NM93, Sim94], control abstraction[GT95], or through training a neural network[dPF93, GTH98].

Deriving the necessary controls to animate a physically-based model may be another area in which both motion planning and libraries of captured motions might be useful. Motion planning problems that involve dynamic constraints (*kinodynamic planning*) can be formulated as a search among the space of available controls to move a physical system from one state to another[DXCR93, BL93, Fer96, LK99, LM99]. Alternatively, one can envision using the "inverse dynamics" of a given physically-based model in order to compute the set of controls necessary to achieve a particular captured motion. Ultimately, libraries of "canned motions" may eventually be replaced by libraries of "canned sets of controls" that can be used in combination with a physically-based model of a character.

Clearly, as the computational resources available to desktop systems grows, increasingly sophisticated physically-based models can be used in a variety of ways in order to generate increasingly realistic animations.

Simulated Sensing

Creating an autonomous animated agent with believable behavior in an interactive virtual environment will ultimately require some kind of simulated sensing (see Chapter 4). This can include one or more of simulated visual, aural, olfactory, or tactile sensing. The basic idea is to more realistically model the flow of information from the virtual environment to the character. The character should act and react according to what it perceives.

Sensory information can be encoded at both a low level and a high level and utilized by high-level decision-making processes of the animated agent. Examples of sensory encodings include "all objects that are currently visible", "all other characters that are currently nearby", or "sounds that can be currently heard". Because animated agents operate in virtual environments, they can potentially avoid many of the problems that physical agents (robots) have when dealing with sensory information (e.g. noisy data, conflicting data, etc). However, in the interests of realism, it may not always be desirable to equip animated agents with perfect sensors. For example, real humans can become confused in rooms with mirrors, or mistake a tiger-skin for a real tiger. In the same way that some computer-generated images of real-life objects are sometimes criticized as being "too perfect", an animated agent that never makes mistakes might also be criticized as being unrealistic.

The output of the agent's sensors can be used to incrementally build a perceived model of the world, with which the agent utilizes to plan its motion. This completes the cycle of the virtual control loop. Note that the agent's perceived model of the virtual world need not coincide with the actual model. This means that agents can sometimes make poor judgments, or be forced to make decisions based on incomplete information (which is what happens to humans in real life).